МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 61.00.00.000 ПЗ

Група ІПм-24-3

Петрів Соломія

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Петрів Соломія Богданівна

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми Software as a Service (SaaS)

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Петрів С.Б.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Корнута Володимир Андрійович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Петрів Соломії Богданівні

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми Software as a Service (SaaS)” керівник проекту (роботи) Корнута Володимир Андрійович, к.т.н., доцент затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проекту (роботи) 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та моделі побудови SaaS-систем, методи забезпечення безпеки та ізоляції даних, алгоритми управління підписками та інтеграції AI-сервісів

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Теоретичні основи SaaS-систем, архітектурні стилі побудови платформ, моделі розгортання та рівні зрілості
2. Методи забезпечення безпеки та ізоляції даних, алгоритми автентифікації та авторизації, методи управління підписками та моделі монетизації
3. Аналіз проблематики та існуючих рішень, розробка архітектури платформи, схеми бази даних та основних алгоритмів системи
4. Програмна реалізація інтерфейсу, серверної логіки, інтеграція зовнішніх сервісів та AI-сервісів, тестування системи, розгортання та монвторинг

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Модель ізоляції на рівні бази даних (рис. 1.1, ст. 29)
2. Модель ізоляції на рівні схеми (рис. 1.2, ст. 30)
3. Модель ізоляції на рівні рядків (рис. 1.3, ст. 31)
4. Схема архітектури для SaaS-платформи (рис. 2.1, ст. 52)
5. Структура бази даних SaaS-платформи для AI-голосового навчання (рис. 2.2, ст. 56)
6. Схема гібридної архітектури інтеграції Vapi (рис. 2.3, ст. 63)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	20.09.2025	виконано
2	Аналіз архітектур SaaS-систем, методів безпеки та освітніх платформ	01.10.2025	виконано
3	Дослідження методів автентифікації, управління підписками та інтеграції AI-сервісів	12.10.2025	виконано
4	Огляд існуючих платформ AI-навчання та формулювання вимог до системи	25.10.2025	виконано
5	Розробка архітектури платформи, схеми бази даних та алгоритмів функціонування	05.11.2025	виконано
6	Програмна реалізація SaaS-платформи, інтеграція сервісів та тестування	22.11.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 90 с., 18 рис., 2 табл., 40 джерел.

Тема: Моделі, методи та алгоритми Software as a Service (SaaS).

Об’єкт дослідження: процеси надання програмного забезпечення як сервісу в хмарних обчислювальних середовищах з інтеграцією AI-технологій для освітніх застосувань.

Мета роботи: удосконалення моделей, методів та розробка алгоритмів побудови SaaS-платформи для AI-голосового навчання з персоналізованими асистентами при забезпеченні ефективної взаємодії в режимі реального часу.

Предмет дослідження: моделі, методи та алгоритми управління ресурсами, забезпечення безпеки даних, контролю доступу та інтеграції AI-сервісів у системах Software as a Service.

Результати дослідження:

Виконано аналіз моделей та методів побудови SaaS-систем, досліджено алгоритми забезпечення ізоляції та безпеки даних. На основі аналізу запропоновано власну архітектурну модель SaaS-платформи для AI-голосового навчання та алгоритми її роботи.

Висновок:

В результаті досліджень отримано моделі, методи та алгоритми побудови SaaS-платформи для AI-голосового навчання, що забезпечують низьку латентність взаємодії, повну ізоляцію даних, автоматичне масштабування та економічну ефективність.

SAAS-ПЛАТФОРМА, AI-ГОЛОСОВЕ НАВЧАННЯ, ПЕРСОНАЛІЗОВАНІ АСИСТЕНТИ, МУЛЬТИТЕНАНТНІСТЬ, ROW-LEVEL SECURITY, БЕЗСЕРВЕРНА АРХІТЕКТУРА, УПРАВЛІННЯ ПІДПИСКАМИ, ІНТЕГРАЦІЯ AI-СЕРВІСІВ, ЛАТЕНТНІСТЬ ВЗАЄМОДІЇ.

ANNOTATION

Master's work: 90 p., 18 fig., 2 tab., 40 sources.

Topic: Models, Methods, and Algorithms of Software as a Service (SaaS).

Object of research: the processes of providing software as a service in cloud computing environments with the integration of AI technologies for educational applications.

Purpose: to improve models, methods, and develop algorithms for building a SaaS platform for AI voice-based learning with personalized assistants while ensuring efficient real-time interaction.

Subject of research: models, methods, and algorithms for resource management, data security, access control, and integration of AI services in Software as a Service systems.

Research results:

An analysis of models and methods for building SaaS systems was carried out, and algorithms for ensuring data isolation and security were investigated. Based on the analysis, a custom architectural model of a SaaS platform for AI voice-based learning and algorithms for its operation were proposed.

Conclusion:

As a result of the research, models, methods, and algorithms for building a SaaS platform for AI voice-based learning were obtained, providing low interaction latency, complete data isolation, automatic scaling, and cost efficiency.

SAAS PLATFORM, AI VOICE-BASED LEARNING, PERSONALIZED ASSISTANTS, MULTITENANCY, ROW-LEVEL SECURITY, SERVERLESS ARCHITECTURE, SUBSCRIPTION MANAGEMENT, AI SERVICE INTEGRATION, INTERACTION LATENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SaaS – Software as a Service (програмне забезпечення як послуга)

IaaS – Infrastructure as a Service (інфраструктура як послуга)

PaaS – Platform as a Service (платформа як послуга)

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (прикладний програмний інтерфейс)

REST – Representational State Transfer (передача репрезентативного стану)

WebRTC – Web Real-Time Communication (веб-комунікація в реальному часі)

RLS – Row-Level Security (безпека на рівні рядків)

JWT – JSON Web Token (веб-токен JSON)

OAuth – Open Authorization (відкрита авторизація)

TLS – Transport Layer Security (безпека транспортного рівня)

HTTPS – Hypertext Transfer Protocol Secure (протокол безпечної передачі гіпертексту)

AES – Advanced Encryption Standard (розширений стандарт шифрування)

SQL – Structured Query Language (мова структурованих запитів)

CDN – Content Delivery Network (мережа доставки контенту)

SDK – Software Development Kit (набір засобів розробки)

CI/CD – Continuous Integration/Continuous Deployment (безперервна інтеграція/розгортання)

СУБД – система управління базами даних

ПЗ – програмне забезпечення

ВСТУП

Актуальність роботи

Сучасний розвиток інформаційних технологій характеризується інтенсивним переходом від традиційних моделей розгортання програмного забезпечення до хмарних сервісів. Модель Software as a Service (SaaS) стала домінуючою парадигмою надання програмного забезпечення, що забезпечує користувачам доступ до застосунків через мережу Інтернет без необхідності локального встановлення та обслуговування. За даними аналітичних агентств, ринок SaaS демонструє стабільне зростання з річним темпом понад 18%, а глобальний ринок освітніх технологій (EdTech), включно з SaaS-рішеннями, продовжує стрімко розширюватися та оцінюється в сотні мільярдів доларів, що підтверджує актуальність досліджень у цій сфері.

Перехід до моделі SaaS супроводжується численними технічними та організаційними викликами, які потребують розробки нових моделей, методів та алгоритмів. Серед ключових проблем виділяються забезпечення масштабованості системи при зростанні кількості користувачів, гарантування безпеки та ізоляції даних в умовах мультитенантної архітектури, ефективне управління підписками та контроль доступу до функцій залежно від тарифного плану, оптимізація використання обчислювальних ресурсів через автоматичне масштабування, інтеграція зовнішніх спеціалізованих сервісів для розширення функціональності.

Особливої актуальності набувають дослідження моделей та методів побудови SaaS-платформ у освітній сфері, де зростає попит на персоналізоване навчання з використанням штучного інтелекту. Інтеграція AI-технологій, зокрема великих мовних моделей, систем розпізнавання та синтезу мови, у освітні SaaS-платформи відкриває нові можливості для створення інтерактивних навчальних систем з голосовою взаємодією. Проте створення таких систем вимагає вирішення специфічних задач щодо забезпечення низької латентності взаємодії в реальному часі, персоналізації AI-асистентів для різних предметів, ефективної архітектури інтеграції з множиною зовнішніх AI-сервісів.

Критичний аналіз існуючих рішень показує, що наявні моделі та методи часто орієнтовані на вирішення окремих задач, таких як балансування навантаження або забезпечення відмовостійкості, без комплексного врахування специфіки SaaS-архітектури у поєднанні з вимогами реалтайм AI-взаємодії.

Існуючі освітні платформи з AI-технологіями мають суттєві обмеження що унеможливлюють їх використання для комплексного голосового навчання: частина з них зосереджена лише на мовній тематиці чи окремих навчальних сценаріях, інші – підтримують голосову взаємодію, але не забезпечують персоналізацію, створення власних AI-асистентів чи структуроване збереження навчальних сесій. Для деяких платформ характерною є відсутність гнучких механізмів налаштування голосу, стилю комунікації та предметної області, а також недостатня підтримка індивідуальних навчальних траєкторій, що є критично важливим для комплексного навчання.

Таким чином, аналіз демонструє, що жодне з існуючих рішень не забезпечує комплексного підходу, який поєднує голосову взаємодію в реальному часі для різних предметних областей, можливість створення користувачами персоналізованих AI-асистентів з налаштуванням голосу, стилю комунікації, тематики та тривалості сесій, структуроване зберігання історії навчальних діалогів, диференційовану монетизацію через тарифні плани з обмеженням функціональності. Тому розробка нових моделей, методів та алгоритмів для SaaS-систем з AI-голосовою взаємодією, що усувають виявлені обмеження, є актуальним науковим завданням, яке має важливе практичне значення для розвитку вітчизняної EdTech-індустрії.

Зв'язок роботи з науковими програмами, планами, темами

Магістерська робота виконана в рамках наукових досліджень кафедри, присвячених розробці інтелектуальних систем управління хмарними обчислювальними ресурсами, методів забезпечення якості надання хмарних сервісів та застосування технологій штучного інтелекту у освітніх системах. Тематика роботи узгоджується з пріоритетними напрямами розвитку науки і техніки України в галузі інформаційних та комунікаційних технологій, зокрема з напрямом «Інформаційно-комунікаційні технології та системи». Дослідження також відповідає стратегії

цифрової трансформації України та національній програмі розвитку цифрової економіки, що передбачає впровадження хмарних технологій та штучного інтелекту у різні сфери суспільного життя, включаючи освіту.

Мета і задачі дослідження

Метою роботи є удосконалення моделей, методів та алгоритмів побудови SaaS-платформи, зокрема для AI-голосового навчання з персоналізованими асистентами, що забезпечує ефективну взаємодію в режимі реального часу при дотриманні вимог до безпеки даних, масштабованості та економічної ефективності.

Розроблена платформа повинна забезпечувати низьку латентність голосової взаємодії, повну ізоляцію даних між користувачами, автоматичне масштабування під навантаженням та бути економічно вигідною через модель оплати за фактичне використання ресурсів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Провести аналіз існуючих моделей, методів та архітектур SaaS-систем, виявити їх переваги та недоліки для застосування у освітніх платформах з AI-взаємодією.
2. Дослідити методи забезпечення безпеки та ізоляції даних у SaaS-системах, алгоритми автентифікації та авторизації, методи управління підписками та алгоритми розподілу ресурсів.
3. Розробити архітектурну модель SaaS-платформи для AI-голосового навчання з урахуванням вимог до масштабованості, мультитенантності, безпеки даних та інтеграції з зовнішніми AI-сервісами.
4. Забезпечити метод ізоляції даних користувачів на основі Row-Level Security з інтеграцією JWT-токенів автентифікації.
5. Запропонувати метод гібридної інтеграції голосових AI-сервісів у SaaS-платформу для мінімізації латентності.
6. Реалізувати програмне забезпечення SaaS-платформи для AI-голосового навчання як практичну апробацію розроблених моделей, методів та алгоритмів.
7. Провести експериментальну верифікацію розроблених рішень через функціональне тестування, тестування продуктивності та тестування безпеки.

Об'єктом дослідження є процеси надання програмного забезпечення як сервісу в хмарних обчислювальних середовищах з інтеграцією AI-технологій для освітніх застосувань.

Предметом дослідження є моделі, методи та алгоритми управління ресурсами, забезпечення безпеки даних, контролю доступу та інтеграції AI-сервісів у системах Software as a Service.

Методи дослідження. Для досягнення мети роботи використовуються методи системного аналізу предметної області SaaS-систем та освітніх платформ, методи проектування архітектури програмних систем та баз даних, методи криптографії та безпеки для механізмів захисту даних, технології веб-розробки та інтеграції з зовнішніми AI-сервісами, методи тестування програмного забезпечення.

Наукова новизна одержаних результатів

Здійснено аналіз моделей SaaS-архітектур, методів забезпечення безпеки та ізоляції даних, алгоритмів управління підписками та способів інтеграції AI-сервісів у освітні платформи, наслідком якого було отримано нову архітектурну модель, алгоритм контролю доступу на основі метаданих підписки та метод гібридної інтеграції голосових AI-сервісів з мінімізацією латентності взаємодії.

Практичне значення одержаних результатів

На основі проведеного дослідження було розроблено повнофункціональну SaaS-платформу для AI-голосового навчання з персоналізованими асистентами, яка може бути впроваджена у освітній процес для індивідуального навчання студентів, корпоративного навчання співробітників, підготовки до іспитів.

Особистий внесок здобувача

- Проведено порівняльну характеристику існуючих платформ AI-навчання та виявлено їх обмеження.

- Розроблено архітектурну модель SaaS-платформи, метод ізоляції даних через Row-Level Security, алгоритми управління підписками та контролю доступу, метод гібридної інтеграції голосових AI-сервісів.
- Реалізовано повнофункціональну SaaS-платформу з клієнтським інтерфейсом, серверною логікою, інтеграціями з зовнішніми сервісами та механізмами безпеки.
- Проведено експериментальну верифікацію через функціональне тестування, тестування продуктивності та безпеки з аналізом результатів.

Структура та обсяг роботи

Магістерська робота викладена на 90 сторінках друкованого тексту, який складається з вступу, чотирьох розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 17 рисунків, 2 таблиці.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА МЕТОДИ ПОБУДОВИ SAAS-СИСТЕМ

1.1 Еволюція хмарних обчислень та концептуальні основи SaaS

1.1.1 Історія розвитку хмарних технологій

Ідея хмарних обчислень бере свій початок ще у 1960-х роках, коли з розвитком концепції обчислювальних мереж та розподілених систем з'явилася можливість віддаленого доступу до обчислювальних ресурсів. Одним із перших, хто висловив думку про «комп'ютинг як утиліту», був Джон Маккарті, який припускав, що у майбутньому обчислювальні потужності надаватимуться користувачам подібно до електроенергії чи водопостачання. У 1970-х роках із розвитком мейнфреймів і систем з поділом часу (time-sharing systems) сформувались перші технічні передумови для реалізації цієї ідеї: користувачі могли отримувати доступ до обчислювальних ресурсів через термінали, не володіючи власним апаратним забезпеченням.

Подальший розвиток інформаційних технологій у 1980–1990-х роках був пов'язаний із масовим поширенням персональних комп'ютерів та клієнт-серверних архітектур. У цей період акцент змістився від централізованих обчислень до локальних мереж, де дані і програми зберігалися безпосередньо на пристроях користувачів. Проте швидке зростання обсягів даних і потреб у взаємодії між різними системами знову поставило питання про централізацію ресурсів та ефективне управління ними.

Наприкінці 1990-х років із розвитком інтернету та веб-технологій з'явилися перші сервіси, які можна вважати попередниками сучасних хмарних рішень. Компанія Salesforce у 1999 році представила модель постачання програмного забезпечення через веб-браузер, що стало першим практичним втіленням концепції Software as a Service (SaaS). Ця ідея дозволила клієнтам використовувати складні бізнес-додатки без необхідності встановлення чи підтримки власної інфраструктури. У 2000-х роках до ринку приєдналися великі компанії, такі як Amazon, Google і Microsoft, які створили власні хмарні платформи – Amazon Web Services (AWS),

Google Cloud Platform (GCP) та Microsoft Azure. Вони запропонували користувачам масштабовані ресурси, які можна орендувати залежно від потреб, що поклало початок епосі масового розвитку хмарних обчислень.

Сьогодні хмарні технології є невід'ємною частиною цифрової економіки. Вони забезпечують основу для сучасних IT-послуг, штучного інтелекту, великих даних, Інтернету речей та інших інноваційних напрямів. Еволюція хмарних систем продовжується у напрямі підвищення гнучкості, безпеки та автоматизації, а також у бік гібридних та мультихмарних рішень, які поєднують переваги різних постачальників і типів інфраструктури.

1.1.2 Моделі надання хмарних сервісів (IaaS, PaaS, SaaS)

Залежно від рівня абстракції та ступеня управління ресурсами, хмарні обчислення поділяються на кілька основних моделей надання сервісів: інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). Ці моделі відображають різний рівень відповідальності між користувачем і постачальником сервісу (рис. 1.1).

Модель IaaS (Infrastructure as a Service) передбачає надання користувачеві базових обчислювальних ресурсів – віртуальних машин, сховищ, мережевих компонентів. Користувач самостійно встановлює і налаштовує операційні системи, прикладне програмне забезпечення та інші компоненти. Основною перевагою IaaS є висока гнучкість та можливість масштабування ресурсів відповідно до навантаження. Класичними прикладами є Amazon EC2, Microsoft Azure Virtual Machines та Google Compute Engine.

Модель PaaS (Platform as a Service) забезпечує користувача готовим середовищем для розробки, тестування та розгортання програмних продуктів. Постачальник платформи бере на себе управління інфраструктурою, операційними системами та середовищем виконання, дозволяючи розробникам зосередитися на створенні функціоналу. Відомими представниками цієї моделі є Google App Engine, Microsoft Azure App Service, Heroku. Основна мета PaaS – спростити розробку програм та скоротити час їх виходу на ринок.

Найвищим рівнем є SaaS (Software as a Service), коли користувач отримує готовий додаток, який повністю обслуговується постачальником. Усі технічні аспекти – сервери, бази даних, оновлення та безпека – залишаються на стороні провайдера, а користувач взаємодіє лише з інтерфейсом через браузер чи мобільний застосунок. Прикладами SaaS є такі сервіси, як Google Workspace, Microsoft 365, Slack або Zoom. Ця модель стала найбільш поширеною серед бізнес-користувачів завдяки низькому порогу входу, передбачуваним витратам і високій доступності.

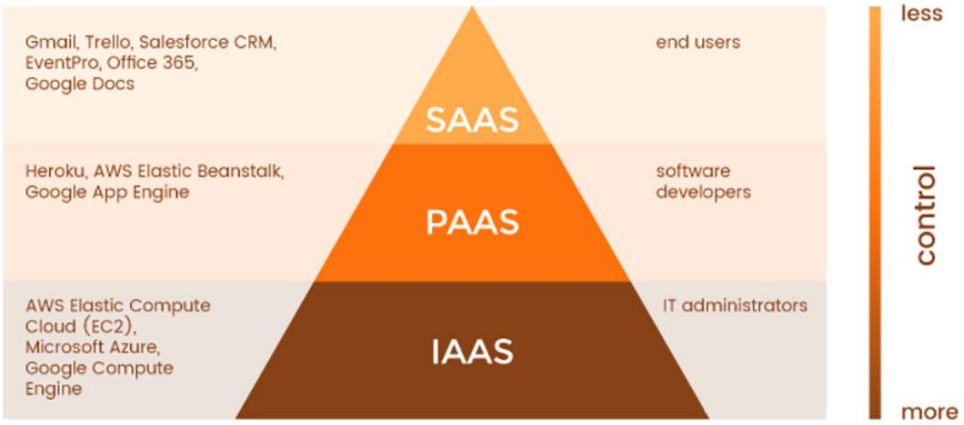

Рис. 1.1. Моделі надання хмарних сервісів

Окрім класичних моделей IaaS, PaaS та SaaS, у хмарних обчисленнях виділяють також інші підходи до надання сервісів. Наприклад, Function as a Service (FaaS) або безсерверні обчислення дозволяють запускати окремі функції або блоки коду без необхідності керувати інфраструктурою, сплачуючи лише за фактичне використання ресурсів. Backend as a Service (BaaS) забезпечує готові серверні рішення для мобільних та веб-додатків, включно з автентифікацією, базами даних та API, спрощуючи розробку фронтенду. Ці моделі розширюють можливості хмарних платформ і дозволяють компаніям обирати оптимальні сервіси відповідно до специфіки бізнесу та вимог до управління ресурсами.

Таким чином, три моделі хмарних сервісів утворюють ієрархію, де кожна наступна забезпечує вищий рівень абстракції, але меншу гнучкість налаштування. Обрання моделі залежить від потреб користувача, типу застосування та вимог до контролю, безпеки й масштабування.

1.1.3 Ключові характеристики SaaS-моделі

Модель Software as a Service (SaaS) забезпечує користувачам доступ до програмного забезпечення через Інтернет, усуваючи потребу в локальній інсталяції та управлінні складною IT-інфраструктурою. Її популярність зростає завдяки поєднанню зручності, економічної ефективності та гнучкості, що робить SaaS привабливим як для малих, так і для великих компаній.

Ключові характеристики SaaS-моделі:

- Доступність через Інтернет – користувач може працювати із застосунком у будь-який час і з будь-якого пристрою з підключенням до мережі, що забезпечує мобільність і зручність, особливо в умовах віддаленої роботи.
- Централізоване управління – всі оновлення, виправлення помилок та вдосконалення виконуються провайдером автоматично, без участі користувачів, що знижує адміністративні витрати та підвищує стабільність системи.
- Масштабованість – система легко адаптується до змін кількості користувачів або навантаження; гнучкі тарифи дозволяють оплачувати лише реально використані ресурси.
- Економічна ефективність – відсутність потреби у придбанні ліцензій, серверного обладнання та власної інфраструктури зменшує початкові витрати; передбачувані абонентські платежі знижують витрати на техпідтримку та персонал.
- Безпека та надійність зберігання даних – багаторівневі механізми захисту, резервного копіювання та шифрування забезпечують високий рівень безпеки, часто вищий, ніж у власних системах підприємств.
- Швидке впровадження та інтеграція – легка інтеграція з іншими системами через API, доступ до нових функцій без додаткових витрат, можливість аналітики в реальному часі.

Отже, SaaS-модель поєднує зручність, доступність і економічну вигоду, дозволяючи підприємствам ефективно використовувати ресурси та зосередитися на основній діяльності. Подальший розвиток SaaS спрямований на підвищення персоналізації, автоматизації та інтеграції з іншими хмарними та AI-технологіями, що робить її ключовою складовою сучасних цифрових екосистем.

1.2 Огляд предметних областей застосування SaaS

1.2.1 CRM, ERP, Collaboration tools

Однією з найважливіших сфер застосування моделей SaaS є корпоративні інформаційні системи, які забезпечують управління клієнтськими відносинами, внутрішніми ресурсами та взаємодію між працівниками. Найпоширенішими серед них є CRM, ERP та інструменти для колаборації, які забезпечують централізоване управління клієнтами, ресурсами та командною взаємодією.

CRM-системи (Customer Relationship Management) у форматі SaaS дозволяють компаніям централізовано керувати взаємовідносинами з клієнтами без необхідності встановлення програмного забезпечення на локальних серверах. Класичні CRM традиційно потребували значних інвестицій у інфраструктуру та персонал для підтримки, тоді як SaaS-підхід надає повний функціонал через вебінтерфейс за моделлю підписки. Такі платформи, як Salesforce, HubSpot чи Zoho CRM, стали прикладами успішної реалізації CRM як хмарного сервісу. Вони пропонують інструменти для збору, обробки й аналізу інформації про клієнтів, а також засоби автоматизації продажів, маркетингу та підтримки.

Основними перевагами SaaS-CRM є можливість швидкої інтеграції з іншими корпоративними системами, гнучкість налаштувань під потреби компанії та централізоване оновлення функціоналу. Дані оновлюються в режимі реального часу, забезпечуючи синхронну роботу команд продажів, маркетингу й підтримки. Вбудовані аналітичні модулі дозволяють визначати поведінкові патерни клієнтів і формувати персоналізовані пропозиції, що підвищує якість сервісу та лояльність.

ERP-системи (Enterprise Resource Planning) у моделі SaaS відіграють ключову роль у цифровій трансформації підприємств. Якщо раніше впровадження ERP було надзвичайно дорогим і тривалим процесом, то хмарна модель дозволяє зменшити початкові витрати та скоротити час запуску. SaaS-ERP об’єднують різні бізнес-процеси – від фінансового обліку й управління персоналом до контролю запасів і логістики – у єдиному середовищі. Такий підхід полегшує моніторинг ефективності та сприяє прийняттю обґрунтованих управлінських рішень.

Однією з ключових переваг SaaS-ERP є можливість динамічного масштабування відповідно до змін бізнес-потреб. Компанії можуть обирати лише необхідні модулі, поступово розширюючи функціонал без ризику технічної несумісності. Хмарні ERP також забезпечують автоматичне оновлення системи, що гарантує відповідність сучасним стандартам безпеки та вимогам регуляторів. У таких рішеннях, як SAP S/4HANA Cloud, Oracle Fusion Cloud ERP або Microsoft Dynamics 365, реалізовано широкий набір інструментів аналітики, планування ресурсів та управління ланцюгами постачання, що дозволяє підприємствам будь-якого розміру підвищувати продуктивність і гнучкість.

Суттєвою перевагою SaaS-моделі для ERP є також підвищена доступність сервісу, що особливо важливо для міжнародних корпорацій і компаній із розподіленими офісами. Крім того, постачальники SaaS рішень активно впроваджують інструменти машинного навчання та штучного інтелекту, які допомагають автоматизувати рутинні операції, прогнозувати фінансові показники та зменшувати людський фактор у процесах прийняття рішень.

Ще однією значною сферою застосування SaaS є колаборaційні інструменти – платформи, які забезпечують спільну роботу команд у реальному часі. Ця категорія сервісів набула особливого значення після глобального переходу до віддаленої та гібридної роботи. Collaboration tools включають хмарні рішення для обміну повідомленнями, відеоконференцій, керування завданнями, спільного редагування документів і зберігання файлів. Найвідомішими прикладами є Slack, Microsoft Teams, Google Workspace, Zoom, Trello та Asana.

Основною цінністю колаборaційних інструментів SaaS є забезпечення безперервності бізнес-процесів навіть за умов віддаленого доступу. Завдяки централізованому зберіганню інформації й автоматичній синхронізації даних усі учасники команди мають актуальну інформацію в режимі реального часу. Такі сервіси також забезпечують високу безпеку, оскільки всі дані передаються в зашифрованому вигляді, а доступ до них контролюється через систему ролей.

Таким чином, CRM, ERP і Collaboration tools як ключові сфери використання SaaS формують основу сучасного корпоративного IT-ландшафту, сприяючи цифровій

трансформації бізнесу, підвищенню ефективності процесів і зміцненню конкурентоспроможності компаній.

1.2.2 Освітні платформи як приклад SaaS

Сфера освіти є ще однією важливою областю застосування SaaS, де хмарні технології сприяють розширенню доступу до знань, підвищенню якості навчального процесу та оптимізації управління освітніми установами. Такі рішення стали особливо актуальними у період масового переходу на дистанційне навчання, оскільки вони забезпечили безперервність освітнього процесу в умовах обмеженого фізичного доступу до навчальних закладів.

Хмарні освітні сервіси, такі як Google Workspace for Education, Moodle Cloud або Coursera for Business, надають інфраструктуру для створення, поширення та оцінювання навчального контенту. Завдяки SaaS-моделі навчальні заклади можуть користуватися сучасними IT-рішеннями без необхідності розгортання власних серверів і технічного персоналу для їх обслуговування. Викладачі отримують можливість створювати інтерактивні курси, проводити онлайн-заняття, організовувати тести та відстежувати успішність студентів. Студенти, своєю чергою, мають постійний доступ до навчальних матеріалів із будь-якого пристрою.

Однією з головних переваг SaaS у сфері освіти є адаптивність систем до різних форматів навчання. Платформи підтримують змішане, дистанційне й індивідуальне навчання, що дозволяє враховувати потреби різних категорій слухачів. Крім того, SaaS-підхід забезпечує централізоване оновлення контенту, швидке масштабування та безпечне зберігання даних студентів і викладачів. Хмарні технології також сприяють розвитку аналітики навчання: на основі даних про активність студентів системи формують рекомендації щодо покращення освітніх програм і підвищення ефективності викладання.

Сучасні SaaS-платформи для освіти часто інтегрують елементи штучного інтелекту, які дозволяють персоналізувати навчальний процес. Наприклад, системи можуть автоматично підбирати завдання залежно від рівня знань студента, виявляти труднощі у засвоєнні матеріалу та пропонувати додаткові ресурси. Це підвищує

мотивацію до навчання й ефективність освітнього процесу загалом. Також SaaS-модель спрощує взаємодію між студентами та викладачами через форуми, чати, відеозустрічі й спільну роботу над проєктами.

Окрім університетів і шкіл, SaaS-рішення активно впроваджуються в корпоративному секторі для організації внутрішнього навчання персоналу. Такі платформи, як TalentLMS, Docebo або SAP SuccessFactors Learning, дозволяють створювати систему безперервного професійного розвитку, відстежувати прогрес співробітників і формувати індивідуальні траєкторії навчання. Таким чином, SaaS у сфері освіти не лише змінює підхід до навчання, а й виступає важливим інструментом розвитку людського капіталу.

У підсумку можна зазначити, що сфери застосування SaaS надзвичайно різноманітні, однак саме CRM, ERP, Collaboration tools і освітні платформи найбільш яскраво демонструють переваги цієї моделі. Вони відображають здатність SaaS масштабуватись, адаптуватись до потреб користувачів і забезпечувати ефективність у різних галузях, від бізнесу до освіти. Усі ці приклади підтверджують, що SaaS став ключовим чинником цифрової еволюції, змінюючи способи роботи, навчання та взаємодії у сучасному суспільстві.

1.3 Архітектурні моделі побудови SaaS-систем

1.3.1 Архітектурні стилі та підходи до проєктування SaaS

Архітектурні стилі визначають фундаментальні принципи організації програмного забезпечення, способи взаємодії компонентів і підходи до масштабування та розвитку системи. У контексті SaaS-платформ правильний вибір архітектури є критично важливим, оскільки він впливає на продуктивність, надійність, безпеку та гнучкість платформи. SaaS-системи можуть будуватися за різними підходами, серед яких найбільш поширеними є монолітна архітектура, сервісно-орієнтована архітектура, мікросервіси та подієво-орієнтована архітектура.

Монолітна архітектура є класичним підходом, при якому всі компоненти додатка (інтерфейс користувача, бізнес-логіка та доступ до даних) об’єднані в єдину

кодову базу і розгортаються як один блок. Така організація робить розробку та тестування досить простими, особливо для невеликих команд, і дозволяє швидко запускати проєкт із мінімальними витратами на інфраструктуру. Водночас монолітні системи мають обмежені можливості масштабування, оскільки збільшення навантаження вимагає масштабування всього додатка, а будь-яка зміна потребує повного повторного розгортання. Тому монолітні архітектури найчастіше застосовуються для невеликих SaaS-додатків, прототипів та стартапів на початкових етапах розвитку.

Сервісно-орієнтована архітектура (SOA) розглядає систему як набір незалежних сервісів, кожен із яких відповідає за конкретну бізнес-функцію. Сервіси взаємодіють через стандартизовані протоколи обміну повідомленнями, що дозволяє ізолювати помилки одного сервісу та масштабувати лише ті компоненти, які потребують більшої потужності. SOA підвищує гнучкість системи, полегшує інтеграцію з іншими корпоративними додатками і дозволяє повторно використовувати сервіси у різних рішеннях. Проте цей підхід вимагає складнішого управління, ретельного проєктування сервісів і забезпечення надійної міжсервісної комунікації.

Мікросервісна архітектура, яка часто розглядається як розвиток ідей SOA, поділяє додаток на невеликі автономні сервіси, кожен із яких реалізує конкретну бізнес-можливість і має власну базу даних та життєвий цикл розгортання. Така архітектура забезпечує максимальну гнучкість і масштабованість, дозволяє незалежно оновлювати та масштабувати окремі частини системи і значно полегшує ізоляцію відмов. Водночас вона вимагає розвинених практик DevOps, автоматизованого моніторингу та ефективного управління мережею, оскільки кількість компонентів і точок взаємодії значно зростає.

Подієво-орієнтована архітектура (EDA) є підходом, у якому взаємодія між компонентами відбувається через події, що генеруються та обробляються асинхронно. Така організація дозволяє створювати високорівневі реактивні системи, які легко масштабуються та швидко реагують на зміни даних у реальному часі. Цей стиль особливо підходить для платформ з високим навантаженням і динамічними

бізнес-процесами, таких як електронна комерція, IoT-системи та потокові сервіси. Водночас він потребує надійного управління подіями та ретельного контролю за їхньою послідовністю, щоб забезпечити коректність роботи системи.

Еволюція архітектурних підходів, від моноліту до мікросервісних і подієво-орієнтованих моделей, демонструє загальну тенденцію до дедалі глибшої декомпозиції та відокремлення інфраструктурних завдань від прикладної логіки. Ця трансформація поступово привела до появи моделей безсерверних обчислень, зокрема Function as a Service, у межах яких виконання коду здійснюється у формі невеликих незалежних функцій, що активуються у відповідь на події або зовнішні запити. Такі підходи виявляються особливо ефективними у випадках нерівномірних, пікових або короткотривалих навантажень, а також для виконання фонових чи допоміжних обчислювальних процесів.

У сучасній практиці розробки SaaS-систем дедалі частіше застосовується гібридний підхід, у якому поєднуються різні архітектурні моделі. Одним із можливих сценаріїв є використання базового монолітного ядра, що відповідає за ключову функціональність платформи, у поєднанні з окремими безсерверними компонентами, які виконують окремі завдання. Така комбінація може бути досить ефективною на практиці, оскільки дає змогу зберегти цілісність і відносну простоту моноліту для стабільних частин системи, водночас отримуючи гнучкість і економічність Serverless-моделей там, де це справді доцільно.

Вибір конкретного стилю залежить від масштабів системи, вимог до продуктивності, складності бізнес-логіки, ресурсів команди та стратегічних цілей компанії. Розуміння особливостей кожного архітектурного підходу дозволяє розробникам і архітекторам обирати оптимальні рішення для побудови ефективних, надійних та масштабованих SaaS-систем.

1.3.2 Моделі розгортання SaaS

Моделі розгортання SaaS визначають організацію роботи платформи для користувачів, розподіл ресурсів, зберігання даних, виконання бізнес-логіки та доступ до інтерфейсу. Вибір моделі впливає на безпеку, масштабованість, ефективність

використання ресурсів і можливість адаптації системи під індивідуальні потреби клієнтів. Основними підходами є single-tenant та multi-tenant моделі, які відрізняються способом організації ізоляції користувачів та керування спільними ресурсами.

У single-tenant моделі розгортання кожен клієнт (тенант) отримує власний екземпляр програмного забезпечення, бази даних та інфраструктури. Це означає, що кожна організація працює в ізольованому середовищі, де зберігаються її дані, конфігурації та користувачі. Такий підхід часто застосовують організації з високими вимогами до безпеки, конфіденційності або індивідуального налаштування. Наприклад, великі фінансові установи або державні організації можуть використовувати single-tenant модель, оскільки вона гарантує повну ізоляцію даних та дозволяє впроваджувати власні політики безпеки.

Однією з головних переваг single-tenant моделі є високий рівень контролю над системою, адже замовник може самостійно визначати графік оновлень і інтеграцій, використовувати окремі версії продукту, а ізольоване середовище полегшує дотримання нормативних вимог і обмежує ризики технічних проблем або кібератак лише межами конкретного тенанта. Водночас ця модель має недоліки – підтримка окремих інстансів потребує більше ресурсів, підвищує витрати на інфраструктуру та адміністрування, ускладнює оновлення і обмежує масштабування, оскільки додавання нових клієнтів вимагає створення нових інстансів, що може впливати на продуктивність і швидкість розгортання.

На противагу цьому, multi-tenant модель розгортання дозволяє одночасне використання одного екземпляра додатка кількома клієнтами. Користувачі працюють у спільному середовищі, але їхні дані залишаються логічно ізольованими. Цей підхід оптимізує використання обчислювальних ресурсів, зменшує витрати на обслуговування та забезпечує централізоване управління. Multi-tenant моделі застосовуються у більшості сучасних SaaS-рішень, таких як Microsoft 365 або Slack.

Основною перевагою multi-tenant моделі є економія масштабу, оскільки спільне використання інфраструктури дозволяє обслуговувати більшу кількість клієнтів із меншими витратами, а оновлення функцій і виправлення помилок відбувається одночасно для всіх користувачів. Централізоване адміністрування

спрощує контроль безпеки, моніторинг продуктивності та впровадження політик доступу. Водночас multi-tenant модель вимагає ретельного підходу до ізоляції даних і безпеки. Оскільки всі користувачі працюють у спільному середовищі, необхідно впроваджувати механізми, які гарантують, що дані одного клієнта не будуть доступні іншому.

У сучасних SaaS-рішеннях часто використовують гібридні моделі розгортання, які поєднують елементи single-tenant і multi-tenant моделей. Наприклад, додаток може мати спільний рівень бізнес-логіки для всіх клієнтів, але окремі бази даних для зберігання їхніх даних. Такий компроміс забезпечує баланс між безпекою та ефективністю, що особливо важливо для корпоративних систем.

Підсумовуючи, вибір моделі розгортання залежить від цілей, масштабу системи, вимог до безпеки, продуктивності та гнучкості. Single-tenant моделі підходять для клієнтів із високими вимогами до ізоляції та контролю, тоді як multi-tenant моделі забезпечують оптимізацію витрат і масштабованість, необхідну для масових SaaS-продуктів. Моделі розгортання SaaS визначають не лише технічну ефективність, але й бізнес-модель продукту, адже саме вони формують баланс між індивідуальністю, витратами і швидкістю розвитку.

1.3.3 Рівні зрілості SaaS

Розвиток рішень типу SaaS можна розглядати через призму рівнів зрілості (рис. 1.2), які показують, як змінюється структура та організація програмного забезпечення, а також його здатність масштабуватися і адаптуватися до потреб клієнтів. Ці рівні дозволяють оцінити, наскільки платформа ефективно використовує ресурси, наскільки гнучкою є конфігурація для окремих користувачів і як організовано обслуговування та безпеку даних.

На першому рівні, Custom Tenant, кожен клієнт отримує свій власний екземпляр додатка, розміщений на виділених ресурсах. Спільними залишаються лише базові інфраструктурні компоненти. Такий підхід нагадує ранні моделі Application Server і дозволяє клієнту мати повний контроль над своїм середовищем. З іншого боку, обслуговування кожного екземпляра окремо є складним і трудомістким,

а будь-яке оновлення або виправлення помилок потрібно впроваджувати для кожного клієнта окремо. Попри це, перший рівень дозволяє зменшити витрати на апаратне забезпечення і спрощує адаптацію традиційних додатків до моделі SaaS.

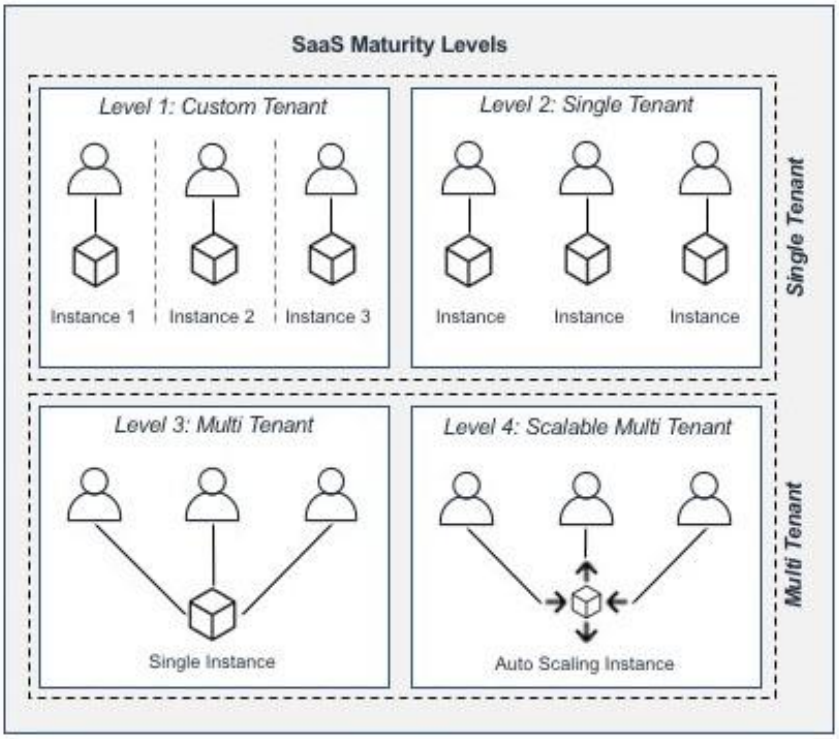

Рис. 1.2. Рівні зрілості SaaS

Другий рівень зрілості, Configurable Applications, передбачає спільну кодову базу для всіх клієнтів, проте кожен отримує окремий екземпляр додатка. Завдяки цьому постачальник може пропонувати широкий спектр налаштувань та конфігурацій, дозволяючи адаптувати інтерфейс і функціональність під потреби конкретного користувача. Перевага такого підходу полягає в тому, що будь-яке оновлення коду автоматично стає доступним усім клієнтам, що значно зменшує витрати часу та ресурсів на підтримку системи. Водночас для одночасної роботи багатьох інстансів потрібні значні апаратні ресурси, що збільшує витрати на інфраструктуру.

Третій рівень зрілості, Configurable Applications and Multi-Tenant, демонструє перехід до більш економного і масштабованого підходу. Один екземпляр додатка обслуговує всіх клієнтів, а конфігураційні метадані дозволяють забезпечити

персоналізований інтерфейс і набір функцій для кожного користувача. Дані залишаються конфіденційними завдяки політикам безпеки, а використання одного додатка для всіх клієнтів дозволяє значно економити обчислювальні ресурси. Основне обмеження цього рівня полягає у масштабованості, яка залежить від потужності сервера.

Четвертий рівень, Configurable, Multi-Tenant and Scalable Applications, характеризується повною масштабованістю. Постачальник SaaS керує великою кількістю клієнтів на серверній фермі з балансуванням навантаження, що дозволяє збільшувати кількість користувачів і серверів без модифікації самого додатка. Виправлення помилок і оновлення функціональності застосовуються для всіх клієнтів одночасно, що робить підтримку системи більш ефективною та прогнозованою. Цей рівень забезпечує оптимальний баланс між ефективністю використання ресурсів, безпекою та гнучкістю.

Сьогодні ринок SaaS демонструє поєднання елементів single-tenant і multi-tenant моделей, що дозволяє одночасно забезпечувати індивідуальні налаштування для великих корпоративних клієнтів та ефективне використання ресурсів для масових продуктів. Корпоративні користувачі очікують інтеграції з власними додатками, можливості контролю оновлень та конфігурацій, а також гнучкі моделі підписки, що поєднують ліцензійні та підписні схеми.

Таким чином, рівні зрілості SaaS допомагають зрозуміти, як платформи розвиваються від простих індивідуальних рішень до високоефективних, масштабованих і універсальних систем, здатних задовольнити потреби користувачів різного масштабу.

1.4 Методи забезпечення безпеки SaaS-платформ

1.4.1 Моделі ізоляції даних (Database, Schema, Row Level)

Безпека та ізоляція даних є одними з ключових викликів при розробці мультитенантних SaaS-платформ. Оскільки ресурси системи спільно використовуються багатьма клієнтами, існує ризик несанкціонованого доступу,

витоку або пошкодження інформації. Кожен орендар очікує, що його дані будуть повністю відокремлені від інших користувачів, навіть якщо вони фізично зберігаються на спільній інфраструктурі. Для досягнення цієї мети використовуються різні моделі ізоляції даних, які визначають рівень поділу інформації між клієнтами: на рівні бази даних, схеми або окремих рядків.

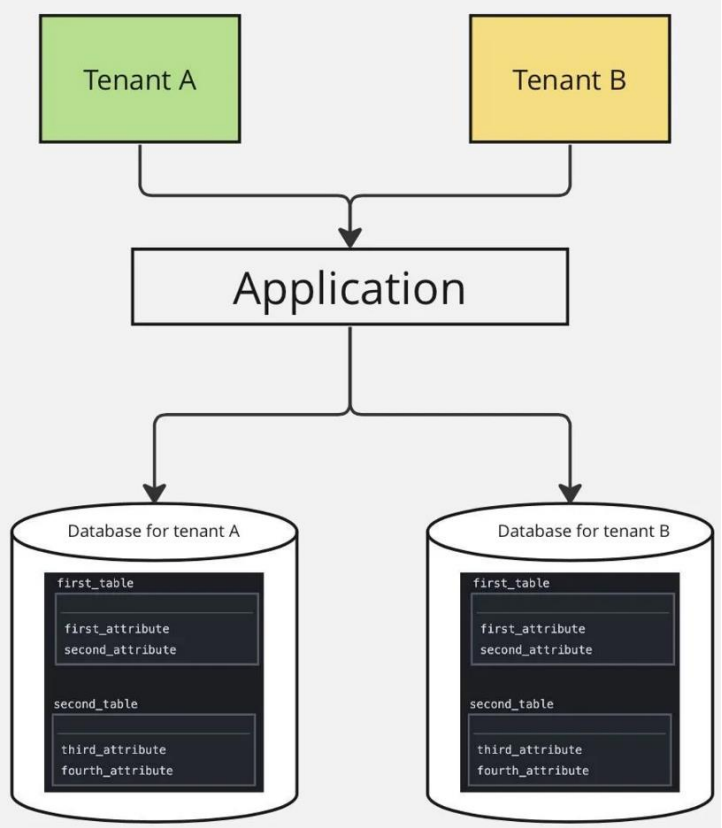

Рис. 1.3. Модель ізоляції на рівні бази даних

Модель ізоляції на рівні бази даних (Database-per-Tenant), що зображена на рисунку 1.3, передбачає створення окремої фізичної бази для кожного клієнта. Така організація забезпечує максимальний рівень безпеки та незалежності даних: кожна база має власні схеми, таблиці, індекси та транзакції. Основними перевагами є простота резервного копіювання та відновлення даних, можливість індивідуального масштабування ресурсів для великих клієнтів і гарантія того, що компрометація одного орендаря не вплине на інших. Недоліком цього підходу є високі витрати на інфраструктуру та обслуговування, складність адміністрування при великій кількості клієнтів та обмежена ефективність використання ресурсів.

Модель ізоляції на рівні схеми (Schema-per-Tenant), що зображена на рисунку 1.4, реалізується в межах однієї фізичної бази, де для кожного клієнта створюється окрема схема з таблицями, індексами та іншими об’єктами. Цей підхід забезпечує певний рівень безпеки при більш ефективному використанні ресурсів. Адміністрування спрощується, оскільки оновлення програмної логіки можна проводити централізовано. Масштабування здійснюється шляхом додавання нових схем без розгортання окремих баз. Основні ризики пов’язані з необхідністю суворого контролю доступу до схем та потенційним ефектом «галасливого сусіда», коли активний клієнт може уповільнити роботу інших.

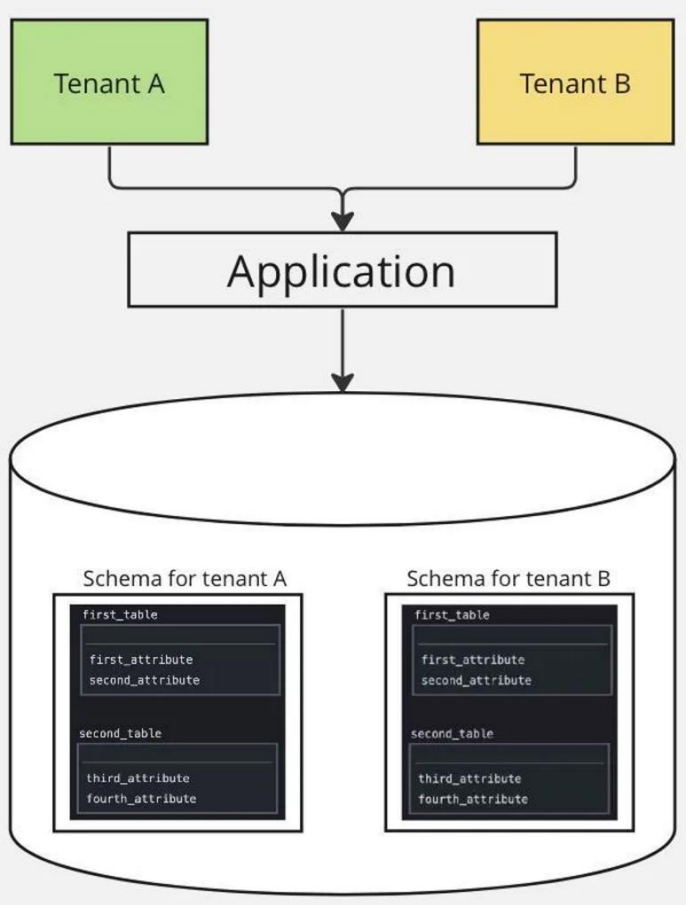

Рис. 1.4. Модель ізоляції на рівні схеми

Модель ізоляції на рівні рядків (Row-Level Isolation), що зображена на рисунку 1.5, передбачає спільне використання бази та схеми, де дані всіх клієнтів зберігаються у тих самих таблицях, але кожен запис маркується унікальним tenant ID. Доступ до даних контролюється на рівні запитів і прикладної логіки, що забезпечує

логічну, але не фізичну ізоляцію. Цей підхід дозволяє обслуговувати велику кількість користувачів з мінімальними витратами на інфраструктуру і простим горизонтальним масштабуванням. Продуктивність запитів залишається високою завдяки ефективному індексуванню та кешуванню. Основні ризики пов’язані з можливим витоком даних при неправильному налаштуванні політик безпеки та ускладненням аудиту та моніторингу.

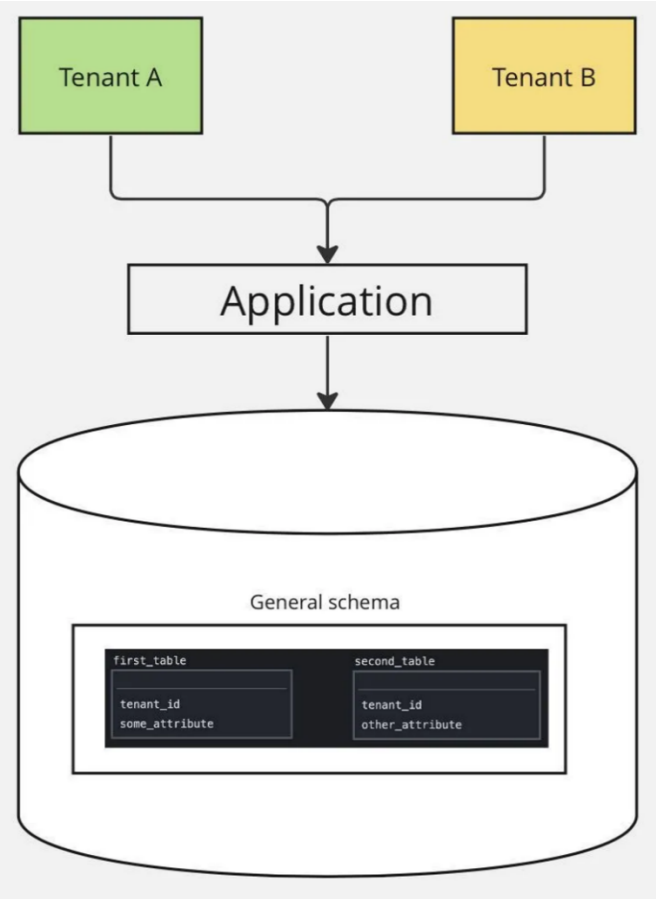

Рис. 1.5. Модель ізоляції на рівні рядків

Вибір конкретної моделі ізоляції залежить від кількості клієнтів, критичності даних, вимог до продуктивності та наявних ресурсів платформи. Для великих корпоративних клієнтів із високими вимогами до безпеки зазвичай використовують Database-per-Tenant. Для середніх клієнтів ефективною є Schema-per-Tenant, де дані ізольовані на логічному рівні, але з економією ресурсів. Для масових користувачів застосовується Row-Level Isolation, що забезпечує масштабованість і високу ефективність використання інфраструктури.

Сучасні SaaS-платформи дедалі частіше використовують гібридні моделі, поєднуючи різні підходи відповідно до категорії клієнтів і типу даних. Наприклад, Salesforce Einstein Analytics обслуговує понад 150,000 організацій за триступеневою гібридною схемою: enterprise-клієнти ізольовані на рівні баз, professional-tier користувачі – на рівні схем, а стандартні користувачі працюють із RLI у спільній базі. AWS Aurora Serverless дозволяє динамічно створювати окремі бази для великих клієнтів і використовувати спільні інстанси з RLI для масових користувачів, оптимізуючи витрати та забезпечуючи автоматичне масштабування ресурсів.

Таким чином, модель ізоляції даних є критично важливим рішенням, що визначає баланс між безпекою, продуктивністю, масштабованістю та економічною ефективністю. Більшість SaaS-платформ починають із Row-Level Isolation, а з ростом числа клієнтів і складності системи поступово переходять до гібридної моделі, що дозволяє підтримувати високий рівень безпеки та задоволеності користувачів.

1.4.2 Механізми Row-Level Security

Row-Level Security (RLS) є сучасним механізмом контролю доступу на рівні бази даних, який забезпечує ізоляцію даних шляхом автоматичного фільтрування рядків у запитах на основі визначених політик безпеки. На відміну від традиційних методів авторизації, що реалізуються на рівні застосунку, RLS забезпечує захист даних безпосередньо у системі управління базою даних, що зменшує ризики помилок у коді програми та підвищує надійність системи.

Механізм RLS функціонує шляхом автоматичного додавання умов фільтрації до SQL-запитів, включаючи операції SELECT, INSERT, UPDATE та DELETE. Кожен запит перевіряє політики безпеки і дозволяє доступ лише до тих рядків, які відповідають встановленим правилам. У контексті SaaS-платформ типовою політикою є перевірка відповідності ідентифікатора орендаря (tenant_id) ідентифікатору поточного автентифікованого користувача, що дозволяє гарантувати, що кожен користувач бачить лише свої дані.

Важливо розрізняти поняття Row-Level Security та Row-Level Isolation (RLI). RLI є архітектурною моделлю організації мультитенантності в SaaS-системах, яка

описує логічне розділення даних кількох орендарів у спільних таблицях за допомогою полів-дискримінаторів. RLS, у свою чергу, є технічним механізмом, реалізованим у СУБД (PostgreSQL, Oracle,), що дозволяє на практиці реалізувати таку ізоляцію.

RLS надає низку переваг для SaaS-платформ. По-перше, він дозволяє централізовано управляти політиками безпеки та застосовувати їх автоматично до всіх запитів. По-друге, механізм підтримує використання різних політик для різних операцій з однією таблицею, що забезпечує гнучкість і масштабованість. По-третє, RLS дозволяє забезпечити відповідність законодавчим вимогам щодо захисту персональних даних (наприклад, GDPR), налаштовуючи доступ на рівні користувача або регіону.

Водночас RLS потребує продуманого проєктування для забезпечення продуктивності системи. Кожен запит із застосуванням політики виконує додаткову перевірку, що може впливати на швидкість обробки даних при високих навантаженнях. Тому важливо оптимізувати індексацію полів, що беруть участь у фільтрації, а також планувати ефективне адміністрування політик у межах DevOps-процесів. У складних або великих SaaS-рішеннях доцільно поєднувати RLS із фізичним розділенням баз або схем для великих клієнтів, що дозволяє досягти балансу між продуктивністю та безпекою.

Таким чином, Row-Level Security є потужним механізмом реалізації ізоляції даних у багатокористувацьких SaaS-платформах. Він забезпечує централізоване управління політиками доступу, підвищує безпеку і прозорість роботи системи та зменшує складність програмного коду. У поєднанні з архітектурним підходом RLI та іншими методами контролю доступу, RLS дозволяє створювати масштабовані та безпечні SaaS-рішення для роботи з великими обсягами клієнтських даних.

1.4.3 Криптографічні методи захисту інформації

Криптографія відіграє ключову роль у забезпеченні конфіденційності, цілісності та автентичності даних у SaaS-системах. У багатокористувацькому середовищі, де дані різних клієнтів передаються і зберігаються на спільних серверах, застосування криптографічних методів є обов’язковим елементом безпеки. Основна

мета шифрування полягає у тому, щоб навіть у випадку несанкціонованого доступу до бази даних або каналів зв’язку зловмисник не міг інтерпретувати вміст інформації.

Шифрування даних у SaaS-середовищі реалізується на кількох рівнях: під час зберігання (data-at-rest), під час передавання (data-in-transit) та під час обробки (data-in-use). Для зберігання даних зазвичай застосовуються симетричні алгоритми, такі як AES (Advanced Encryption Standard), які забезпечують високу швидкість шифрування та низьке навантаження на систему. Для передавання інформації через мережу використовується протокол TLS (Transport Layer Security), що гарантує захист від перехоплення і модифікації даних під час їх руху між клієнтом і сервером.

Додатковим засобом підвищення безпеки є використання асиметричних алгоритмів шифрування, таких як RSA або Elliptic Curve Cryptography (ECC), для управління ключами. Завдяки цьому зменшується ризик компрометації головного ключа, адже публічні ключі можна вільно розповсюджувати, а приватні зберігаються лише на захищених серверах. Крім того, у SaaS-платформах часто застосовується концепція «bring your own key» (BYOK), коли клієнт має можливість самостійно керувати своїми ключами шифрування, що підвищує довіру до сервісу.

Важливу роль відіграють і механізми гешування. Для зберігання паролів або перевірки цілісності файлів використовуються алгоритми SHA-2 або SHA-3. Гешування забезпечує односторонню трансформацію даних, унеможливлюючи відновлення оригінального значення, що значно ускладнює несанкціонований доступ. Для додаткового захисту рекомендується використовувати «salt» – випадкові дані, що додаються перед гешуванням, щоб запобігти атакам типу «rainbow tables».

Крім традиційних криптографічних засобів, SaaS-системи дедалі частіше застосовують методи шифрування з нульовим розкриттям (Zero-Knowledge Encryption) та гомоморфне шифрування. Перший дозволяє перевіряти достовірність операцій без необхідності розкриття самих даних, а другий – виконувати обчислення над зашифрованою інформацією, не розшифровуючи її.

Належна реалізація криптографічних методів потребує постійного моніторингу, оновлення алгоритмів та перевірки сертифікатів безпеки. Вразливості у криптографічних бібліотеках або неправильне управління ключами можуть

призвести до катастрофічних наслідків. Тому провайдери SaaS зазвичай впроваджують системи управління ключами (KMS) та аудиторські механізми, що фіксують усі операції з шифруванням і розшифруванням.

1.4.4 Алгоритми автентифікації та авторизації

У сучасних SaaS-платформах механізми автентифікації та авторизації є критично важливими для забезпечення безпеки, масштабованості та надійності роботи системи. Поширення хмарних сервісів сприяло розвитку різноманітних підходів до управління доступом, серед яких виділяють токен-орієнтовані методи, класичні сесійні механізми та протоколи делегованого доступу.

JSON Web Token (JWT) є компактним самодостатнім токеном, що складається з заголовка, тіла (payload) та криптографічного підпису, які кодуються у форматі Base64URL. У тілі містяться твердження про користувача, час життя токена та інші метадані, необхідні для авторизації. Ключовою перевагою JWT є можливість автономної верифікації кожним мікросервісом без звернення до центрального сховища, що спрощує горизонтальне масштабування та підвищує ефективність розподілених архітектур.

Водночас самодостатність JWT створює складнощі у реалізації механізму примусового завершення сесії. Поки токен не прострочений і є валідним, сервер не може його відкликати без допоміжних засобів, таких як чорні списки або трекінг активних токенів. Тому безпечне керування життєвим циклом токенів передбачає використання короткочасних access token та довготривалих refresh token, які дозволяють оновлювати пару токенів після додаткової перевірки.

Класичний підхід session management застосовується тоді, коли необхідний високий рівень керованості: відстеження активних сесій, можливість негайного завершення доступу, обмеження паралельних підключень тощо. Після автентифікації сервер створює унікальний ідентифікатор сесії та пов’язує його з даними користувача, що зберігаються у серверному сховищі чи кеші. Клієнт отримує такий ідентифікатор у вигляді cookie, і всі подальші запити прив’язані до відповідного стану на сервері. На відміну від JWT, сесія не містить даних користувача у відкритому

вигляді, що спрощує гнучке управління доступом, але ускладнює масштабування через необхідність синхронізації стану між серверами.

Порівняльний аналіз показує, що JWT оптимально підходить для масштабованих розподілених SaaS-платформ і систем, орієнтованих на інтенсивні API-взаємодії, оскільки не потребує централізованого зберігання стану та легко інтегрується в мікросервісні середовища. Session management, своєю чергою, доцільний у платформах, де важливі гнучкий контроль над сесіями, можливість негайного їх завершення та деталізований моніторинг активності. У практиці розробки SaaS часто застосовуються гібридні моделі: JWT для авторизації API, а сесії для керування сеансами в адміністративній панелі або веб-інтерфейсі.

У SaaS-середовищах, де численні сервіси інтегруються між собою та зі сторонніми додатками, необхідні механізми безпечного делегування доступу без передачі конфіденційних облікових даних. Для цього застосовують протоколи OAuth 2.0 та OpenID Connect (OIDC), які стали стандартом у більшості сучасних веб- і хмарних платформ. OAuth 2.0 забезпечує авторизацію, тоді як OIDC доповнює його механізмами автентифікації.

OAuth 2.0 реалізує модель делегованого доступу, у якій користувач може надати одному сервісу право діяти від його імені на іншому сервісі без передачі пароля. Основою роботи OAuth є видача access token, який визначає обсяг і тривалість дозволеного доступу. OAuth 2.0 підтримує кілька потоків авторизації, серед яких найпоширенішим у SaaS є Authorization Code Flow – найбезпечніший механізм для веб-додатків, що передбачає серверний обмін коду авторизації на токени. Інші потоки, такі як Implicit Flow, нині вважаються застарілими через нижчий рівень безпеки, а Client Credentials Flow застосовується для серверних інтеграцій без участі користувача.

Оскільки OAuth 2.0 не визначає процедуру встановлення особи користувача, для цього розроблено OpenID Connect (OIDC) – протокол автентифікації, що працює поверх OAuth 2.0. OIDC вводить ID Token, який містить дані про автентифікованого користувача: унікальний ідентифікатор, час видачі та інформацію про провайдера. На відміну від access token, ID Token призначений саме для підтвердження особи.

Особливо цінною для SaaS-платформ є підтримка федеративної автентифікації OIDC, яка дозволяє об’єднувати корпоративні системи з хмарними сервісами. Користувач може увійти до SaaS-додатку через Microsoft Entra ID або Keycloak, автоматично отримуючи права доступу відповідно до корпоративних політик. Такий підхід спрощує адміністрування, зменшує кількість паролів та забезпечує централізований контроль безпеки.

Таким чином, сучасні SaaS-платформи використовують комбінацію механізмів автентифікації та авторизації: JWT забезпечує масштабованість API-взаємодій, session management надає контроль над користувацькими сесіями, а OAuth 2.0 та OpenID Connect реалізують безпечне делегування доступу та єдиний вхід (SSO). Вибір конкретного підходу визначається архітектурними вимогами, моделлю масштабування та політиками безпеки платформи.

1.5 Дослідження методів управління підписками

1.5.1 Моделі монетизації SaaS

Монетизація є ключовим аспектом розробки SaaS-продуктів, оскільки визначає економічну модель сервісу, стабільність доходів та можливості масштабування. На відміну від традиційного програмного забезпечення, SaaS передбачає доступ за підпискою або фактичним використанням, тому вибір підходу до монетизації має враховувати тип аудиторії, специфіку продукту, витрати на інфраструктуру та ринкове середовище.

Однією з найпоширеніших є передплатна модель, яка ґрунтується на регулярних періодичних платежах. Така модель підтримує створення декількох тарифних планів із різним набором функцій. Основним недоліком є складність залучення нових користувачів, оскільки не всі готові платити до ознайомлення з продуктом, тому така підписка часто впроваджується після формування певної активної аудиторії.

Поряд із традиційною підпискою поширена модель платних послуг, у межах якої основні можливості доступні безкоштовно, а за додаткові функції користувач

сплачує окремо. Модель підходить для сервісів із чітким поділом на базовий та розширений функціонал. Важливим завданням є визначення межі між безкоштовними і платними можливостями, щоб продукт залишався привабливим, але водночас приносив прибуток.

Для ресурсозатратних та багатоопераційних сервісів доцільною є транзакційна модель, у рамках якої користувач платить лише за фактичне використання. Вартість формується залежно від кількості операцій, обсягу запитів чи інших вимірюваних активностей. Такий підхід забезпечує прозорість для клієнтів і справедливий розподіл витрат щодо навантаження на інфраструктуру.

Рис. 1.6. Основні моделі монетизації SaaS

Популярною є також модель freemium, що передбачає можливість безкоштовного використання базової версії продукту та оплату за повний функціонал. Freemium ефективна в сегменті масових продуктів, де важливо швидко залучити широке коло користувачів. Основний виклик полягає в забезпеченні оптимального балансу між цінністю безкоштовної версії та привабливістю платних можливостей.

Рекламна модель базується на доході від показу оголошень, що дозволяє надавати сервіс безкоштовно. Вона є ефективною лише для продуктів із великим

трафіком, оскільки прибуток залежить від кількості переглядів або кліків. Незважаючи на це, така модель часто використовується як допоміжна та може поєднуватися з іншими підходами, формуючи багатоканальну структуру доходів.

Крім класичних моделей, які можна побачити на рисунку 1.6, SaaS-провайдери використовують й інші стратегії. Партнерська модель передбачає отримання прибутку через реферальні програми або інтеграції з іншими компаніями. Непрямі продажі передбачають роботу через реселерів, які просувають продукт серед власних клієнтів, а прямі – здійснюються командою менеджерів безпосередньо з клієнтами. Також використовують роздрібні або веб-продажі, коли продукт просувають через соціальні мережі чи веб-сайт.

Вибір стратегії монетизації визначається специфікою продукту, бізнес-цілями, конкурентним середовищем та поведінкою користувачів. Комбінування методів монетизації дозволяє SaaS-компаніям адаптуватися до різних сегментів ринку і забезпечувати довгострокову фінансову стабільність.

1.5.2 Управління підписками

Управління підписками в SaaS-платформах передбачає структурований процес супроводу користувача протягом усього часу його взаємодії з сервісом. Він охоплює всі етапи від первинної реєстрації до продовження, скасування або повторного залучення підписки, забезпечуючи стабільний дохід і підтримку користувача на кожному кроці. Розуміння життєвого циклу підписки дозволяє компаніям створювати цінність для клієнтів, підвищувати їх залученість та утримання.

Першим етапом є придбання (Acquisition), коли потенційний користувач вперше знайомиться з платформою. Основним завданням цього етапу є перетворення зацікавлених відвідувачів на підписників, для цього застосовують різноманітні маркетингові інструменти, демонструючи цінність продукту, а також забезпечують зручну та швидку процедуру реєстрації.

Наступним кроком є введення в користування (Onboarding). Після оформлення підписки користувач потребує підтримки для швидкого освоєння сервісу, метою є усунення плутанини, формування впевненості користувача та закладання основи для

довгострокових взаємовідносин із платформою. На цьому етапі застосовуються навчальні матеріали, вітальні повідомлення та доступ до каналів.

Залучення (Engagement) є наступним критично важливим етапом, що передбачає підтримку активності користувача та створення постійної цінності продукту. Регулярна взаємодія через оновлення, освітній контент та можливість залишати відгуки допомагає користувачам відчувати значущість своєї участі та підтримує інтерес до сервісу.

Не менш важливим є етап оплати та виставлення рахунків (Billing and Payments). Його основним завданням є забезпечення прозорості фінансових операцій, коректності нарахувань та дотримання встановлених циклів оплати. Гнучкі опції сплати та автоматизовані нагадування дозволяють знизити ймовірність помилок і підвищують довіру користувачів до платформи.

Коли термін підписки наближається до завершення, реалізується етап продовження підписки (Renewal). Своєчасне нагадування користувачам про цінність продукту, чіткі інструкції щодо поновлення та стимулюючі пропозиції, такі як бонуси або знижки, значно підвищують шанси на подовження підписки.

Однак частина користувачів може відмовитися від підписки, що формує етап відтоку або скасування (Churn or Cancellation). Аналіз причин відмови, отриманий через опитування або автоматизовані повідомлення, дозволяє виявити проблемні аспекти сервісу та запровадити заходи для зменшення подальшого відтоку, наприклад, пропонуючи тимчасове призупинення підписки або зміни тарифу.

Навіть після скасування підписки платформа може прагнути до повторної активації (Reactivation) користувачів. Персоналізовані повідомлення, спеціальні пропозиції та демонстрація нових функцій дозволяють заохотити колишніх користувачів повернутися та відновити підписку, нагадуючи їм про цінність сервісу.

Ефективне управління підписками дозволяє підтримувати довгострокові відносини з користувачами, підвищувати їхню задоволеність і забезпечувати стабільний потік доходу для бізнесу. Кожен етап життєвого циклу від придбання до повторної активації є можливістю створити цінність, закріпити лояльність та зменшити ризик відтоку користувачів.

1.6. Алгоритми розподілу ресурсів та масштабування

1.6.1 Квотування ресурсів за типами підписки

У SaaS-середовищі розподіл ресурсів між користувачами та їх ефективне управління є ключовим аспектом забезпечення стабільної роботи платформи. Одним із основних підходів до цього є квотування ресурсів, яке дозволяє регламентувати обсяг доступних обчислювальних потужностей, сховищ даних, мережевих запитів та інших компонентів сервісу залежно від типу підписки користувача. Такий підхід забезпечує баланс між ефективністю використання інфраструктури та задоволенням потреб клієнтів, водночас захищаючи платформу від надмірного навантаження або зловживання ресурсами.

Моделі квотування формуються на основі тарифних планів, що відрізняються доступом до функцій і обсягами ресурсів. Користувачі базових і безкоштовних планів зазвичай мають нижчі ліміти на кількість API-запитів, швидкість обробки чи обсяги збережених даних, тоді як преміальні плани пропонують розширені квоти та пріоритетний доступ. Це не лише оптимізовує інфраструктуру, а й формує цінову політику, мотивуючи користувачів переходити на вищі рівні підписки.

Важливим аспектом при впровадженні моделей квотування є гнучкість у налаштуванні ресурсів та можливість їх динамічного перегляду. Платформа може динамічно змінювати ліміти, реагуючи на зростання активності або надаючи тимчасові бонуси. Подібні механізми дозволяють збалансувати інтереси користувача та провайдера, підтримуючи стабільність системи без надмірного резервування ресурсів та уникаючи різких обмежень, які можуть погіршити користувацький досвід.

У контексті бізнес-моделі SaaS квотування ресурсів також грає роль у формуванні цінової стратегії та стимулюванні переходу користувачів на вищі плани. Обмеження, встановлені для базових тарифів, мотивують користувачів розширювати підписку, якщо їм потрібні додаткові ресурси, більш швидка обробка або пріоритетний доступ до сервісу. Це поєднує технічні та економічні аспекти управління ресурсами, підвищуючи ефективність роботи платформи і забезпечуючи стабільний дохід для провайдера.

Таким чином, квотування ресурсів за типами підписки є важливим інструментом для оптимізації інфраструктури SaaS-платформи. Воно дозволяє ефективно контролювати використання обчислювальних потужностей, сховища даних та мережевих ресурсів, забезпечують баланс між потребами користувачів та можливостями сервісу, а також створюють передумови для стійкого фінансового розвитку компанії.

1.6.2 Алгоритми автоматичного масштабування

Автоматичне масштабування є одним із ключових механізмів управління SaaS-платформами, що дозволяє ефективно адаптувати обчислювальні ресурси до динамічних змін навантаження. Воно забезпечує безперервну роботу сервісу, мінімізує ризики простоїв та перевантажень, а також оптимізує витрати на інфраструктуру, оскільки ресурси виділяються тільки тоді, коли це дійсно необхідно. Алгоритми автомасштабування дозволяють платформам SaaS автоматично додавати або зменшувати обчислювальні потужності, пам’ять, сховище та інші ресурси відповідно до фактичного споживання користувачами.

Основним завданням автоматичного масштабування є забезпечення оптимальної продуктивності системи при змінному навантаженні. Це досягається за допомогою алгоритмів, які відстежують ключові метрики платформи, такі як завантаження процесора, використання оперативної пам’яті, кількість запитів або обсяг мережевого трафіку. При досягненні встановлених порогових значень система автоматично приймає рішення про масштабування: додаткові ресурси виділяються у випадку зростання навантаження, а при зниженні активності надлишкові ресурси звільняються. Такий підхід дозволяє підтримувати стабільну роботу платформи навіть під час пікових навантажень або швидких змін активності користувачів.

Сучасні алгоритми автоматичного масштабування можна умовно поділити на горизонтальні та вертикальні. Горизонтальне масштабування передбачає збільшення або зменшення кількості серверів, що обслуговують платформу, що дозволяє рівномірно розподіляти навантаження та уникати перевантаження окремих вузлів. Вертикальне масштабування полягає у збільшенні обчислювальних ресурсів

конкретного сервера, наприклад додавання оперативної пам’яті або процесорних ядер, що забезпечує швидке реагування на збільшення навантаження без необхідності додаткових інстанцій. У більшості SaaS-платформ використовують комбінацію обох підходів, що дозволяє досягти оптимальної продуктивності та економічності.

Ключовим аспектом ефективного авт-масштабування є вибір відповідних порогових значень та правил реагування на зміни навантаження. Алгоритми можуть бути реактивними, що реагують на вже виникле навантаження, або прогнозуючими, які на основі аналізу історичних даних і тенденцій передбачають зростання активності користувачів і заздалегідь виділяють додаткові ресурси.

Важливим аспектом є інтеграція алгоритмів автомасштабування з моніторинговими та аналітичними системами. Постійне відстеження ключових показників продуктивності дозволяє своєчасно реагувати на зміни у використанні платформи та оптимізувати роботу алгоритмів масштабування. Це включає контроль затримок обробки запитів, відслідковування часу відповіді серверів, аналіз пікових періодів активності та прогнозування майбутніх потреб. Такий комплексний підхід дозволяє підтримувати високу якість обслуговування і забезпечує безперервність бізнес-процесів SaaS-платформи.

Завдяки автоматичному масштабуванню SaaS-компанії можуть значно підвищити ефективність використання ресурсів, уникати простоїв і перевантажень, а також оптимізувати витрати на інфраструктуру. Воно забезпечує стабільну роботу сервісу при будь-яких рівнях навантаження та сприяє збереженню високого рівня задоволеності користувачів. Крім того, правильно налаштоване автомасштабування дозволяє платформам ефективно адаптуватися до змін у кількості користувачів та їх поведінки, що робить його невід’ємною складовою сучасних SaaS-рішень.

1.6.3 Балансування навантаження

Балансування навантаження є критично важливим механізмом забезпечення стабільної роботи SaaS-платформ в умовах високої динаміки користувацької активності. Його основна мета полягає у рівномірному розподілі запитів між серверами та обчислювальними вузлами, що дозволяє уникати перевантаження

окремих компонентів, зменшувати затримки у відповіді та підвищувати загальну продуктивність системи.

Балансування навантаження реалізується за допомогою апаратних або програмних рішень, які автоматично спрямовують запити користувачів на доступні сервери. У сучасних SaaS-платформах переважають програмні балансувальники, інтегровані в хмарну інфраструктуру, що забезпечує гнучкість і масштабованість. Алгоритми балансування враховують поточне завантаження серверів, час відповіді, географічне розташування користувачів та пріоритети підписки.

Основні стратегії включають кругове (round-robin) призначення запитів, балансування за доступними ресурсами або мінімальним часом відповіді, а також географічне балансування навантаження. Останній підхід перенаправляє користувачів до найближчих дата-центрів для мінімізації мережевих затримок, що є особливо важливим для глобальних SaaS-сервісів.

Балансування навантаження тісно взаємодіє з алгоритмами автоматичного масштабування. При зростанні навантаження балансувальник спрямовує запити на нові сервери, додані внаслідок автоматичного масштабування, а при зниженні активності оптимізує розподіл і вивільняє надлишкові ресурси. Така взаємодія дозволяє підтримувати стабільну роботу платформи при будь-яких коливаннях користувацької активності, зберігаючи високу доступність сервісу та ефективність використання інфраструктури.

Таким чином, балансування навантаження є невід’ємною складовою інфраструктури SaaS-платформи та забезпечує рівномірний розподіл ресурсів, стабільність роботи сервісу й високу якість обслуговування користувачів.

1.7 Висновки до розділу

У першому розділі проведено комплексний аналіз теоретичних основ та методів побудови SaaS-систем. Досліджено еволюцію хмарних обчислень та визначено ключові моделі надання хмарних сервісів (IaaS, PaaS, SaaS) та їх характеристики. Встановлено, що модель SaaS забезпечує найвищий рівень

абстракції, поєднуючи доступність, масштабованість, економічну ефективність та централізоване управління. Аналіз предметних областей застосування SaaS, зокрема CRM, ERP, колаборaційних інструментів та освітніх платформ, показав перспективність інтеграції AI-технологій у SaaS-рішення для освіти.

Досліджено архітектурні моделі, моделі розгортання й рівні зрілості SaaS-платформ, та встановлено, що сучасні платформи переважно використовують гібридні підходи, поєднуючи переваги різних моделей. Розглянуто методи забезпечення безпеки: моделі ізоляції даних, механізми Row-Level Security, криптографічні методи захисту та алгоритми автентифікації і авторизації. Визначено, що Row-Level Isolation у поєднанні з RLS є оптимальним рішенням для платформ із великою кількістю користувачів, забезпечуючи баланс між безпекою та економічною ефективністю.

Проаналізовано методи управління підписками, включаючи основні моделі монетизації SaaS-продуктів та життєвий цикл підписки користувача. Досліджено алгоритми розподілу ресурсів та масштабування: квотування ресурсів за типами тарифних планів, горизонтальне та вертикальне автомасштабування, а також механізми балансування навантаження. Результати теоретичного аналізу формують методологічну базу для проєктування SaaS-платформи AI-голосового навчання з урахуванням вимог до безпеки, масштабованості та економічної ефективності.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА АРХІТЕКТУРИ SAAS-ПЛАТФОРМИ

2.1 Аналіз проблематики та існуючих рішень

2.1.1 Виклики розробки SaaS-платформ для AI-асистованого навчання

Сучасний ринок освітніх технологій зазнає трансформацію, спричинену стрімким розвитком штучного інтелекту та технологій обробки природної мови. Традиційні підходи до дистанційного навчання, що базуються на відеолекціях, текстових матеріалах та статичних тестах, обмежені у можливості забезпечити персоналізований навчальний досвід. Відтак зростає попит на інтелектуальні системи персонального навчання, здатні адаптуватися до індивідуальних потреб студентів і підтримувати інтерактивну взаємодію в реальному часі.

Проте впровадження AI-асистованого навчання ускладнюється низкою технічних і концептуальних викликів. Особливо складним завданням є впровадження голосової взаємодії в реальному часі. Створення природного діалогу вимагає інтеграції систем розпізнавання мовлення, генерації тексту та синтезу голосу. Затримки понад 2–3 секунди критично знижують ефект живого спілкування. Крім того, необхідні педагогічно обґрунтовані механізми контролю темпу та складності пояснень, а також реагування на нерозуміння користувача.

Якість відповідей AI є критичним аспектом. Генеративні моделі схильні до створення правдоподібних, але хибних тверджень (галюцинацій). Забезпечення точності навчальних результатів вимагає застосування структурованих інструкцій, методів Retrieval-Augmented Generation (RAG), аналізу логів діалогів. Оцінюванню підлягають не лише технічні параметри, а й ефективність навчання: зрозумілість пояснень, рівень засвоєння матеріалу, мотивація студентів.

Конфіденційність голосових даних, що містять унікальні ознаки користувачів (тембр, акцент, емоційний стан), потребує відповідності стандартам приватності при передачі до зовнішніх сервісів. Користувачам має надаватися прозора інформація про обробку даних та можливість їх видалення.

Інфраструктурні витрати голосової AI-платформи зростають пропорційно числу користувачів. Необхідні механізми оптимізації: використання моделей різних класів, кешування відповідей, моніторинг витрат та обмеження надмірного використання ресурсів.

Таким чином, розробка SaaS-платформи для AI-асистованого навчання є комплексним завданням, що вимагає інтеграції передових технологій з урахуванням технічних, педагогічних та економічних аспектів. Успішне рішення забезпечує низьку латентність, персоналізовану взаємодію, безпеку даних та економічно стійку модель функціонування.

2.1.2 Порівняльний аналіз існуючих рішень та методів

Аналіз наявних платформ для AI-асистованого навчання показав, що існуючі рішення орієнтовані на вирішення окремих задач без комплексного врахування специфіки SaaS-архітектури у поєднанні з вимогами реалтайм AI-взаємодії. Основні обмеження провідних освітніх платформ із AI-технологіями:

- Платформа Speak орієнтована виключно на вивчення іноземних мов через голосові діалоги з AI. Забезпечує якісне розпізнавання вимови та корекцію помилок, проте не підтримує інші предметні області (математика, природничі науки, програмування) і не дозволяє користувачам створювати персоналізовані асистенти з налаштуванням параметрів взаємодії.
- ChatGPT Voice від OpenAI надає можливість голосової взаємодії з великою мовною моделлю для обговорення широкого кола тем, включаючи освітні. Він забезпечує природний діалог та якісні пояснення, проте не адаптований для структурованого навчання з персоналізованими асистентами для конкретних предметів, не зберігає історію навчальних сесій окремо від загального чату, не надає можливості створення бібліотеки асистентів із різними налаштуваннями голосу та стилю викладання.
- Khanmigo від Khan Academy інтегрує AI-репетитора на основі GPT-4 для різних предметів (математика, природничі науки, історія), забезпечуючи пояснення концепцій та допомогу у вирішенні задач. Використовує переважно текстову

взаємодію з обмеженою підтримкою голосу, не дозволяє користувачам створювати власні персоналізовані асистенти з налаштуваннями параметрів діалогу, а також зберігає історію взаємодій у загальному форматі без структурування за навчальними сесіями.

- Duolingo Max інтегрує можливості GPT-4 для пояснення граматичних помилок та практики розмовних навичок у вивченні мов. Функціональність обмежена виключно мовною тематикою без можливості розширення на інші предметні області. Відсутня можливість створення персоналізованих асистентів із індивідуальними характеристиками.

Узагальнюючи результати аналізу, слід зазначити, що жодна з розглянутих платформ не забезпечує комплексного підходу, який поєднує голосову взаємодію в реальному часі для різних предметних областей; можливість створення користувачами персоналізованих AI-асистентів із налаштуванням голосу, стилю комунікації, тематики та тривалості сесій; структуроване зберігання історії навчальних діалогів; диференційовану монетизацію через тарифні плани з градацією функціональності.

Виявлені обмеження обґрунтовують актуальність розробки інтегрованих моделей, методів та алгоритмів для SaaS-систем із AI-голосовою взаємодією, що має важливе практичне значення для розвитку вітчизняної EdTech-індустрії.

2.2 Вимоги до системи

2.2.1 Функціональні вимоги

Функціональні вимоги визначають що саме повинна робити система, які можливості надає користувачам, які дії підтримує та які бізнес-процеси реалізує. Вони є критичними, оскільки формують основу логіки платформи, визначають сценарії взаємодії з користувачем та напряму впливають на відповідність продукту поставленим цілям. Якщо функціональні вимоги визначені нечітко, платформа може працювати непослідовно, пропускати необхідні можливості або не задовольняти очікування користувачів.

Функціональні вимоги SaaS-платформи для AI-асистованого навчання:

- FR-1. Управління користувачами та автентифікація. Система повинна забезпечувати повний цикл управління обліковими записами користувачів, включаючи реєстрацію через електронну пошту, автентифікацію з використанням багатофакторної перевірки, інтеграцію з провайдерами OAuth для входу через Google. Користувачі мають можливість управляти своїм профілем, включаючи особисту інформацію, налаштування конфіденційності, мовні преференції.
- FR-2. Створення та налаштування AI-асистентів. Користувачі повинні мати можливість створювати персоналізованих AI-асистентів для навчання, визначаючи ключові параметри: назву асистента, предметну область, тему, голосові характеристики, стиль комунікації, очікувану тривалість навчальної сесії.
- FR-3. Проведення голосових навчальних сесій. Основний функціонал платформи полягає у забезпеченні інтерактивних голосових діалогів між користувачем та AI-асистентом у режимі реального часу. Під час сесії система повинна забезпечувати розпізнавання мови, передачу контексту, генерацію відповідей, синтез голосу, відображення транскрипту. Користувач має можливість керувати сесією: призупиняти та відновлювати діалог, вимикати та вмикати мікрофон, завершувати сесію у будь-який момент.
- FR-4. Управління історією навчальних сесій. Система повинна автоматично зберігати інформацію про всі завершені навчальні сесії, включаючи дату та час проведення, назву використаного AI-асистента, тему навчання, повний транскрипт діалогу між користувачем та асистентом. Історія сесій відображається у персональному профілі користувача з візуалізацією загальної статистики.
- FR-5. Система закладок та обраного. Користувачі повинні мати можливість додавати найбільш корисних AI-асистентів до списку обраних для швидкого доступу. Система забезпечує можливість додавання та видалення закладок одним кліком, окремий розділ профілю для перегляду всіх збережених асистентів.
- FR-6. Пошук та фільтрація AI-асистентів. Система повинна надавати потужні інструменти для пошуку навчальних асистентів у загальній бібліотеці платформи. Користувачі можуть здійснювати текстовий пошук за назвою асистента або темою

навчання з підтримкою часткового співпадіння та пошуку по ключових словах. Результати пошуку відображаються у вигляді сітки карток з основною інформацією про кожного асистента.

- FR-7. Управління підписками та тарифними планами. Платформа повинна підтримувати гнучку систему монетизації з декількома рівнями підписок, що надають різний обсяг функціональних можливостей. Система забезпечує автоматичну інтеграцію з платіжним провайдером для обробки підписок, підтримку різних методів оплати, автоматичне продовження підписок з попереднім повідомленням користувача, можливість зміни тарифного плану, скасування підписки зі збереженням доступу до кінця оплаченого періоду.

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги описують якість роботи системи, тобто те, наскільки швидко вона реагує, наскільки безпечна, надійна, масштабована, зручна та доступна. Це критично важливо, бо навіть найкраща функціональність втрачає цінність, якщо система повільна, нестабільна, небезпечна чи незручна для користування. Нефункціональні вимоги формують очікувані стандарти роботи платформи, визначають технічні обмеження та гарантують відповідність продукту професійним нормам і досвіду користувачів.

Нефункціональні вимоги SaaS-платформи:

- NFR-1. Продуктивність та латентність. Система повинна забезпечувати латентність не більше 2 секунд від завершення фрази користувача до початку синтезованої відповіді асистента у 95% випадків. Завантаження основних сторінок інтерфейсу має відбуватися не довше 2 секунд при стандартному широкосмуговому з'єднанні. Перехід між розділами додатку повинен бути миттєвим завдяки використанню клієнтського роутингу.
- NFR-2. Масштабованість. Архітектура системи повинна підтримувати горизонтальне масштабування для обслуговування зростаючої кількості користувачів без деградації продуктивності. Система має коректно обробляти щонайменше 1000 одночасних голосових сесій на початковому етапі з можливістю

розширення шляхом додавання обчислювальних ресурсів. База даних повинна ефективно працювати з мільйонами записів користувачів, асистентів та історії сесій без суттєвого уповільнення запитів.

- NFR-3. Надійність та доступність. Платформа повинна забезпечувати доступність не менше 99.5% часу. Система має включати механізми автоматичного відновлення після збоїв, резервного копіювання критичних даних з можливістю відновлення у разі катастрофічних відмов. При тимчасовій недоступності окремих AI-сервісів система повинна коректно інформувати користувачів про проблему та пропонувати альтернативні дії.
- NFR-4. Безпека та захист даних. Вся комунікація між клієнтом та сервером повинна здійснюватися по захищених HTTPS-з'єднаннях з використанням сучасних криптографічних протоколів TLS. Паролі користувачів зберігаються виключно у хешованому вигляді з використанням bcrypt або argon2, а персональні дані – з дотриманням вимог GDPR Токени автентифікації мають обмежений термін дії та автоматично оновлюються. Система повинна захищати від типових веб-вразливостей: SQL-ін'єкцій, XSS-атак, CSRF-атак.
- NFR-5. Приватність та ізоляція даних. Система повинна забезпечувати повну ізоляцію даних різних користувачів – жоден користувач не може отримати доступ до AI-асистентів, історії сесій, транскриптів діалогів інших користувачів. Користувачі мають право видалити свій обліковий запис разом з усіма пов'язаними даними, і система повинна забезпечувати повне видалення інформації з усіх компонентів інфраструктури протягом 30 днів.
- NFR-6. Зручність використання (Usability). Інтерфейс платформи повинен бути інтуїтивно зрозумілим для користувачів з базовими навичками роботи з веб-додатками. Критичні дії, такі як початок навчальної сесії, повинні виконуватися не більше ніж у 3 кліки від головної сторінки.
- NFR-7. Моніторинг та відстеження помилок. Система повинна включати комплексний моніторинг всіх компонентів з автоматичним сповіщенням команди розробки про критичні помилки та аномалії у роботі. Всі виключення на клієнті та сервері логуються з детальною інформацією про контекст виникнення для

полегшення діагностики. Інтеграція з сервісом аналітики помилок дозволяє відстежувати частоту різних типів помилок та пріоритизувати виправлення.

- NFR-8. Підтримуваність та розширюваність. Кодова база повинна дотримуватися загальноприйнятих практик чистого коду, мати модульну структуру з чітким розділенням відповідальності між компонентами. Система спроектована з урахуванням майбутніх розширень: додавання нових типів AI-асистентів, інтеграції додаткових AI-провайдерів, впровадження нових моделей монетизації.

2.3 Архітектура системи

2.3.1. Загальна архітектура

Архітектура SaaS-платформи для AI-голосового навчання (рис. 2.1) побудована на основі монолітного підходу з безсерверним виконанням, що оптимально відповідає потребам стартапу на початковій стадії розвитку. Платформа відповідає четвертому рівню зрілості SaaS – Configurable, Multi-Tenant and Scalable Applications, забезпечуючи масштабованість через автоматичне балансування навантаження, мультитенантність з ізоляцією даних та конфігурованість завдяки персоналізованим AI-асистентам для кожного користувача.

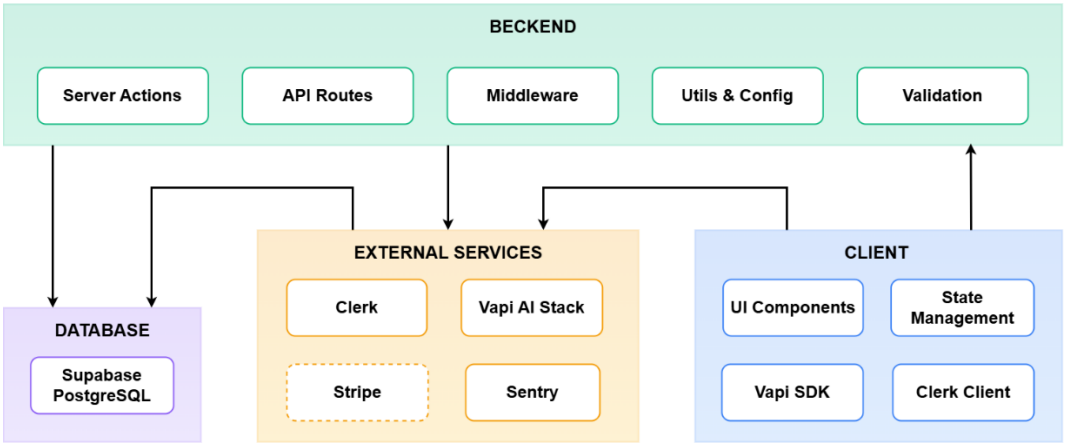

Рис. 2.1. Схема архітектури для SaaS-платформи

Мультитенантність реалізована класичною моделлю, за якої один екземпляр застосунку та спільна база даних обслуговують усіх користувачів. Ізоляція даних

кожного з них забезпечується механізмами RLS, що гарантує незалежність контенту, AI-асистентів, сесій і закладок. Такий підхід знижує операційні витрати та покращує ефективність використання ресурсів порівняно з архітектурою single-tenant.

Вибір монолітної архітектури зумовлений специфікою проєкту та обмеженими ресурсами. У межах єдиної кодової бази інтегровані клієнтський інтерфейс, серверна бізнес-логіка та взаємодія з базою даних, що прискорює вихід продукту на ринок, мінімізує операційні витрати та полегшує налагодження. Потенційні недоліки моноліту, такі як складність масштабування та необхідність повного перерозгортання при внесенні змін, компенсуються використанням безсерверної моделі та модульною структурою коду.

Безсерверне виконання (Serverless) передбачає запуск серверної логіки у форматі окремих функцій, що активуються лише при надходженні HTTP-запитів від клієнта. Це забезпечує автоматичне горизонтальне масштабування, оптимальні витрати в умовах низького навантаження та швидке виділення додаткових ресурсів у пікові періоди, без необхідності ручного керування інфраструктурою.

Інтеграція спеціалізованих зовнішніх сервісів дозволяє делегувати критично важливі функції перевіреним провайдерам. Зовнішні сервіси забезпечують автентифікацію та управління користувачами, фінансові транзакції, обробку голосових даних, зберігання файлів та моніторинг системи. Це зменшує технічні ризики, прискорює розробку, підвищує рівень безпеки та допомагає відповідати індустріальним стандартам.

Підхід Backend-for-Frontend (BFF) реалізований на основі серверних функцій, що виконують роль проміжного шару між клієнтським інтерфейсом і зовнішніми сервісами. Вони здійснюють автентифікацію, авторизацію, валідацію даних, оркестрацію викликів API та обробку помилок, створюючи єдину контрольовану точку бізнес-логіки і безпеки.

Виняток становить інтеграція з Vapi, зокрема під час голосових сесій, використовується гібридна модель: клієнт встановлює пряме підключення до AI-сервісів для зменшення затримки, а конфігурація, запис результатів та логіка керування забезпечуються серверними функціями.

Асинхронна обробка подій реалізована через webhook-механізми, що дають змогу реагувати на зміни у зовнішніх сервісах без постійного опитування. Події обробляються незалежно від активності користувача, без блокування інтерфейсу, з автоматичним повторним виконанням у разі тимчасових збоїв.

У підсумку архітектура поєднує простоту моноліту, економічну ефективність безсерверної моделі, безпеку завдяки використанню зовнішніх сервісів та високий рівень масштабованості, що робить її оптимальною для SaaS-платформи на початковому етапі розвитку. Модульна організація коду забезпечує можливість подальшого виділення високонавантажених компонентів у мікросервіси без радикальних змін системи.

2.3.2 Вибір технологічного стеку

Вибір технологічного стеку для SaaS-платформи AI-голосового навчання є визначальним фактором її ефективності, надійності та масштабованості. Технології мають підтримувати обробку голосових сесій у реальному часі, забезпечувати безпечність коду, інтегруватися з зовнішніми сервісами та надавати сучасний користувацький інтерфейс. Обрана архітектура базується на сучасних веб-технологіях і безсерверній моделі, що дає змогу мінімізувати операційні витрати та спростити масштабування.

Основною мовою програмування є TypeScript, який поєднує гнучкість JavaScript із перевагами статичної типізації. Використання типів підвищує надійність бізнес-логіки та зменшує кількість помилок під час розробки й рефакторингу. Для перевірки коректності структури даних застосовується Zod, що дозволяє використовувати єдине визначення схем як для типів, так і для валідації на етапі виконання. Це особливо важливо для форм конфігурації AI-асистентів, у яких перевіряються параметри сесій, опис теми та інші атрибути.

Фреймворком для розробки веб-додатку є Next.js, який забезпечує поєднання клієнтських і серверних можливостей в одному середовищі. Він підтримує серверний рендеринг, автоматичне розділення коду, оптимізовану маршрутизацію та виконання API-маршрутів без потреби у створенні окремого бекенду. У поєднанні з

інфраструктурою Vercel забезпечується автоматичне безсерверне масштабування й можливість виконання частини обчислень на edge-вузлах для зниження затримки при голосовій взаємодії.

Для роботи з даними застосовується Supabase – керована платформа на основі PostgreSQL. Вона підтримує як реляційні таблиці, так і формат JSON, що дозволяє гнучко зберігати змішані структури даних. Supabase також забезпечує міграції, автоматичні резервні копії та механізми реального часу, що дозволяють оновлювати статистику навчання без перезавантаження сторінки.

Автентифікація й управління користувачами реалізуються через сервіс Clerk, який забезпечує готові компоненти для реєстрації, входу, відновлення доступу та керування профілем. Сервіс автоматично керує сесіями та підтримує диференційований доступ залежно від тарифного плану. Інтеграція Clerk зі Stripe забезпечує синхронізацію підписок у режимі реального часу. Інструмент Stripe Billing автоматизує роботу з рекурентними платежами, податковими розрахунками та інвойсами. Використання Stripe Checkout дозволяє обробляти платіжні дані виключно на стороні платіжного провайдера, значно знижуючи вимоги до безпеки власної інфраструктури. Оновлення стану підписок передаються через вебхуки, забезпечуючи узгодженість між платіжною системою та внутрішньою логікою.

Підсистема голосової взаємодії реалізована за допомогою сервісу Vapi, який об’єднує технології розпізнавання мовлення, генерації відповідей та синтезу голосу в єдиний інтегрований пайплайн. Взаємодія відбувається через WebRTC, що забезпечує мінімальну затримку та підтримку повноцінних діалогових сесій у реальному часі. Конфігурація асистентів здійснюється через промпт-моделі, які визначають стиль, поведінку та навчальний контекст.

Моніторинг платформи здійснюється за допомогою Sentry, який відстежує помилки на клієнтському та серверному рівнях, фіксує стек викликів та іншу діагностичну інформацію. Система сповіщень дає змогу оперативно реагувати на збої, забезпечуючи стабільну роботу платформи у режимі 24/7.

Таким чином, обраний технологічний стек формує комплексну та збалансовану інфраструктуру, що поєднує продуктивність, безпеку, масштабованість

і зручність розробки. Використання TypeScript, Next.js, Supabase, Clerk, Stripe та Vapi забезпечує можливість реалізації голосової взаємодії у реальному часі, підтримку мультитенантності та автоматизацію ключових бізнес-процесів. Такий підхід дозволяє швидко розвивати функціональність платформи та адаптувати її до зростання навантаження й потреб користувачів.

2.3.3 Схема бази даних

Проектування бази даних є ключовим етапом розробки SaaS-платформи для AI-голосового навчання, що забезпечує надійне зберігання, організацію та ефективне управління даними користувачів, AI-асистентів та навчальних сесій. База даних реалізована на PostgreSQL через керовану платформу Supabase, що надає вбудовані механізми безпеки на рівні рядків (Row Level Security), автоматичне резервне копіювання та можливості горизонтального масштабування.

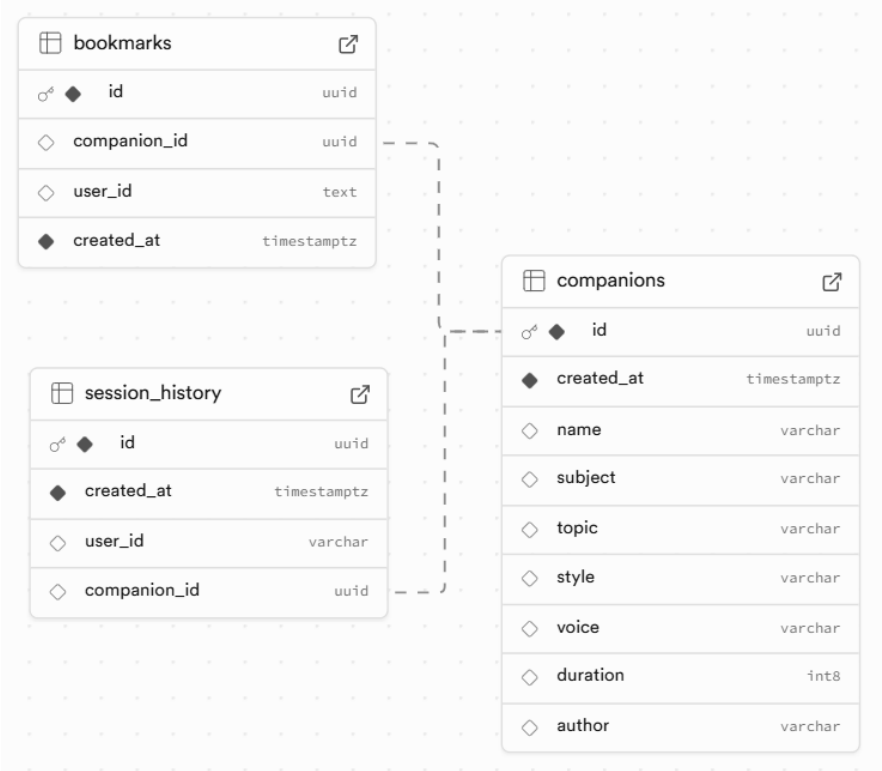

Рис. 2.2. Структура бази даних SaaS-платформи для AI-голосового навчання

Основні таблиці, їх атрибути та логічні зв’язки формують структуру (рис. 2.2), що підтримує створення та налаштування персоналізованих AI-асистентів,

проведення голосових навчальних сесій, відстеження історії та прогресу користувачів, а також організацію закладок. Мінімалістична архітектура з делегованим управлінням користувачами через Clerk спрощує підтримку та знижує операційні ризики.

Ключовою архітектурною особливістю схеми є відсутність окремої таблиці користувачів (users). Управління користувачами повністю делеговано зовнішньому сервісу автентифікації Clerk, що генерує унікальні ідентифікатори. Ці ідентифікатори зберігаються у полях author та user_id відповідних таблиць для встановлення зв'язку даних з конкретними користувачами. Такий підхід забезпечує єдине джерело правди для користувацьких даних, усуває проблеми синхронізації профілів між системами та знижує ризики витоку персональних даних.

Таблиця companions є центральною сутністю системи, що зберігає конфігурацію персоналізованих AI-асистентів. Кожен користувач може створювати необмежену кількість асистентів (або обмежену залежно від тарифного плану), налаштовуючи параметри навчання під конкретні освітні потреби. Поле author забезпечує мультитенантність – всі запити автоматично фільтруються за ідентифікатором поточного користувача. Поля subject та topic дозволяють організувати асистентів за предметними областями та підтримують функціональність пошуку і фільтрації. Налаштування voice та style визначають характеристики голосової взаємодії, які передаються до платформи Vapi для конфігурації синтезу мови та формування інструкцій для мовної моделі.

Таблиця session_history фіксує факти завершених навчальних сесій між користувачами та AI-асистентами. Мінімалістична структура зберігає лише критичні метадані: хто проходив навчання (user_id), з яким асистентом (companion_id) та коли (created_at). Зв'язок з таблицею companions дозволяє відображати повну інформацію про асистента без дублювання даних у кожному записі. Дані історії використовуються для відстеження активності, статистики профілю, аналізу популярності асистентів та предметів.

Дані таблиці взаємопов'язані через логічні зв'язки на основі спільних ідентифікаторів. Зв'язок один-до-багатьох між користувачами та companions

реалізується через поле author – кожен користувач може створити множину асистентів, але кожен асистент має одного автора-власника. Аналогічний зв'язок існує між користувачами та session_history через поле user_id, та між companions та session_history через поле companion_id.

Таблиця bookmarks реалізує функціональність збереження обраних асистентів для швидкого доступу. Структура представляє класичний зв'язок багато-до-багатьох між користувачами та асистентами – один користувач може додати в закладки множину асистентів, а один асистент може бути у закладках множини користувачів.

В подальшому структура може бути розширена додатковими таблицями для зберігання транскриптів діалогів, метрик якості сесій, рейтингів асистентів, спільного доступу до асистентів між користувачами, аналітики навчального прогресу з деталізацією за темами та предметами.

2.3.4 Забезпечення безпеки та ізоляції даних

Забезпечення безпеки та ізоляції даних у SaaS-платформі для AI-голосового навчання реалізовано через багаторівневу систему захисту, що поєднує механізми RLS, криптографічні методи, механізми автентифікації та авторизації, валідацію вхідних даних та моніторинг аномальної активності. Такий підхід обумовлений мультитенантною природою системи з великою кількістю користувачів, де кожен створює власні AI-асистенти, що мають бути повністю ізольовані за умови збереженні економічної ефективності та спільного використання інфраструктури.

Модель Row-Level Isolation обрана як оптимальна для платформи, оскільки забезпечує баланс між безпекою, масштабованістю та економічною ефективністю. Вона зберігає дані всіх користувачів у спільних таблицях з логічною ізоляцією через поля tenant_id. Кожен запис у таблицях companions, session_history та bookmarks маркується полем author або user_id, що містить унікальний ідентифікатор користувача з Clerk.

Row-Level Security у PostgreSQL через Supabase виступає останнім бар'єром безпеки, що захищає від витоку даних навіть у разі помилок у серверному коді або успішних SQL-ін'єкцій. Політики RLS визначають булеві вирази, що автоматично

застосовуються до кожного SQL-запиту, фільтруючи рядки на основі контексту поточного користувача. Для таблиці companions політика SELECT дозволяє читання тільки тих записів, де author збігається з auth.uid(); політика INSERT перевіряє, що новий запис створюється від імені поточного користувача; UPDATE та DELETE обмежені можливістю змінювати лише власні записи. Аналогічні правила застосовуються до session_history і bookmarks.

Криптографічний захист даних реалізується на рівнях data-at-rest, data-in-transit та data-in-use. Дані при зберіганні шифруються засобами інфраструктури Supabase з використанням AES-256. Оскільки система не зберігає критично чутливих персональних даних у власній базі (паролі, email-адреси та профільні дані обробляються Clerk), обсяг даних, що потребує додаткового шифрування, мінімізований.

Передача даних між клієнтом, сервером і базою даних здійснюється виключно через захищені канали. Vercel автоматично забезпечує TLS 1.3 для фронтенду, Supabase підтримує SSL для всіх з’єднань із PostgreSQL, а сторонні інтеграції (Clerk, Stripe, Vapi) використовують HTTPS. Це захищає від атак типу перехоплення трафіку, підміни даних та крадіжки токенів автентифікації.

Валідація вхідних даних здійснюється за допомогою Zod-схем, які перевіряють структуру, типи, обов’язкові поля та відповідність значень допустимим форматам. Поле author завжди встановлюється сервером на основі автентифікованого користувача, і будь-які спроби підміни цього поля зі сторони клієнта ігноруються. Валідація також захищає від SQL-ін’єкцій, XSS та CSRF-атак через параметризовані запити, безпечне рендерування React та використання SameSite cookies.

У результаті сформовано комплексну систему захисту даних, яка забезпечує надійний захист конфіденційності користувачів платформи AI-голосового навчання, об’єднуючи економічну ефективність спільної інфраструктури з високим рівнем логічної ізоляції користувачів, сучасними криптографічними механізмами, надійною автентифікацією, авторизацією та багаторівневим моніторингом. Така архітектура забезпечує стійкість до більшості типових атак і захищає конфіденційні освітні дані навіть у разі часткової компрометації серверних компонентів.

2.4 Розробка алгоритмів роботи системи

2.4.1 Алгоритм реєстрації та автентифікації користувачів

Алгоритм автентифікації платформи побудований на інтеграції з хмарним сервісом Clerk, який забезпечує повний цикл керування доступом та сесіями користувачів. Використання спеціалізованого сервісу дозволяє уникнути складної власної реалізації автентифікації, що є критично важливим для SaaS-середовища. Clerk поєднує модель сесійного керування з підтримкою JWT-токенів і протоколів OAuth 2.0 та OpenID Connect, забезпечуючи єдину точку ідентифікації для всіх компонентів системи.

Процес автентифікації починається з перевірки активної сесії через middleware Next.js. Якщо чинна сесія відсутня, користувач перенаправляється на сторінку авторизації, де доступні методи входу через електронну пошту з паролем або через зовнішніх провайдерів (Google) за протоколами OAuth 2.0/OIDC. Після успішної автентифікації Clerk створює серверну сесію та надсилає на клієнт захищені HttpOnly-cookies з ідентифікатором сесії. Централізоване зберігання сесій на стороні Clerk забезпечує оперативне керування життєвим циклом доступу, включаючи можливість відкликання активних сесій.

На рівні серверних компонентів Next.js застосовуються функції auth() для отримання userId безпосередньо з cookies та currentUser() для завантаження повного профілю користувача з backend Clerk. Для інтеграції з Supabase використовуються короткострокові JWT-токени, що передаються у заголовку Authorization та перевіряються політиками Row Level Security (RLS). Це забезпечує єдиний механізм ідентифікації на всіх рівнях системи.

Оновлення сесій виконується автоматично завдяки серверному керуванню Clerk. При закінченні строку дії сесії або її інвалідації користувач перенаправляється на сторінку входу. Вихід із системи реалізовано через компонент UserButton, який видаляє cookies сесії та блокує доступ до захищених маршрутів. Така архітектура забезпечує баланс між зручністю постійних сесій і високим рівнем безпеки, що відповідає вимогам SaaS-платформ.

2.4.2 Алгоритм управління підписками

Управління підписками реалізовано через інтеграцію Clerk зі Stripe, що забезпечує централізовану обробку тарифних планів і платежів. Оформлення підписки починається зі сторінки тарифних планів, де компонент PricingTable відображає три рівні доступу: безкоштовний, стандартний і преміум. Кожен план містить визначену вартість, перелік функцій та інструменти керування підпискою.

Під час вибору плану Clerk створює checkout-сесію та перенаправляє користувача на хостовану сторінку оплати. Після успішної транзакції метадані користувача оновлюються: встановлюється поле publicMetadata.subscriptionPlan, додаються права доступу, фіксуються дати початку та наступного списання. Метадані доступні на клієнті через useUser(), а на сервері – через currentUser() та auth(), що дозволяє ефективно перевіряти доступ без додаткових запитів до API. У разі невдалого списання здійснюються повторні спроби, а за їх провалом підписка деактивується через два тижні.

Зміна підписки відбувається через customer portal Clerk: апгрейд застосовується миттєво з пропорційним перерахунком вартості, даунгрейд – після завершення поточного оплаченого періоду. Синхронізація статусу підписки виконується автоматично через метадані користувача, що забезпечує консистентність доступу та спрощує архітектуру. Такий підхід автоматизує повний життєвий цикл монетизації, мінімізує кількість коду й гарантує надійну обробку платежів за допомогою індустріального стандарту Stripe.

2.4.3 Алгоритм контролю доступу до функцій

Алгоритм контролю доступу забезпечує обмеження функціональних можливостей на основі тарифного плану користувача через серверну функцію newCompanionPermissions(). Перевірка виконується у три етапи:

- визначення преміум-статусу методом has({ plan: 'pro' }), який для преміум-користувачів дозволяє необмежене створення асистентів;
- отримання ліміту для користувачів нижчих планів через has({ feature: ... }) – три асистенти для безкоштовного плану та десять для стандартного;

- підрахунок фактичної кількості створених асистентів через запит до бази даних.

Фінальна перевірка відбувається через порівняння companionCount із лімітом: якщо companionCount ≥ limit, функція повертає false, блокуючи створення нового асистента. Результат визначає поведінку клієнтського інтерфейсу: при false відображається пропозиція оновити тарифний план, при true – відкривається форма створення асистента.

Для захисту від обходу клієнтських обмежень перевірка повторно виконується на серверному рівні під час створення запису в базі. Функція createCompanion() викликає newCompanionPermissions() та повертає помилку «Subscription required» у разі порушення правил доступу. Алгоритм автоматично адаптується до змін тарифних планів, оскільки метадані підписки оновлюються миттєво. Це забезпечує справедливу монетизацію та запобігає зловживанням.

2.4.4 Алгоритм інтеграції з Vapi AI для голосових сесій

Для забезпечення голосової взаємодії користувачів із персоналізованими AI-асистентами у режимі реального часу розроблено метод гібридної архітектури інтеграції Vapi, що поєднує централізований контроль персоналізації та безпеки з мінімальною латентністю передачі аудіоданих.

Проаналізовано три альтернативні підходи до інтеграції голосових AI-сервісів:

1. Повністю серверна архітектура передбачає проксіювання всіх аудіоданих через власний backend, що забезпечує максимальний контроль, але збільшує латентність на 500–800 мілісекунд через додатковий мережевий hop, руйнуючи природність діалогу.
2. Повністю клієнтська архітектура реалізує пряме WebRTC-з’єднання з фіксованою конфігурацією асистента, що забезпечує мінімальну латентність 1–2 секунди, але унеможливлює персоналізацію на основі даних користувача та створює ризики безпеки через експонування AI-промптів у клієнтському коді.
3. Запропонований гібридний підхід розподіляє відповідальність між серверною та клієнтською частинами відповідно до характеру операцій, що проілюстровано на рисунку 2.3.

Серверна функція configureAssistant() генерує персоналізовану конфігурацію AI-асистента на основі параметрів companion: предмет, тема, стиль комунікації та тип голосу. Функція формує об’єкт CreateAssistantDTO з налаштуваннями транскрибера Deepgram Nova-3 з автоматичною пунктуацією, голосу ElevenLabs, обраного зі структури voices, а також системного промпту GPT-4, який визначає роль асистента як тьютора з інструкціями дотримуватися теми та стилю.

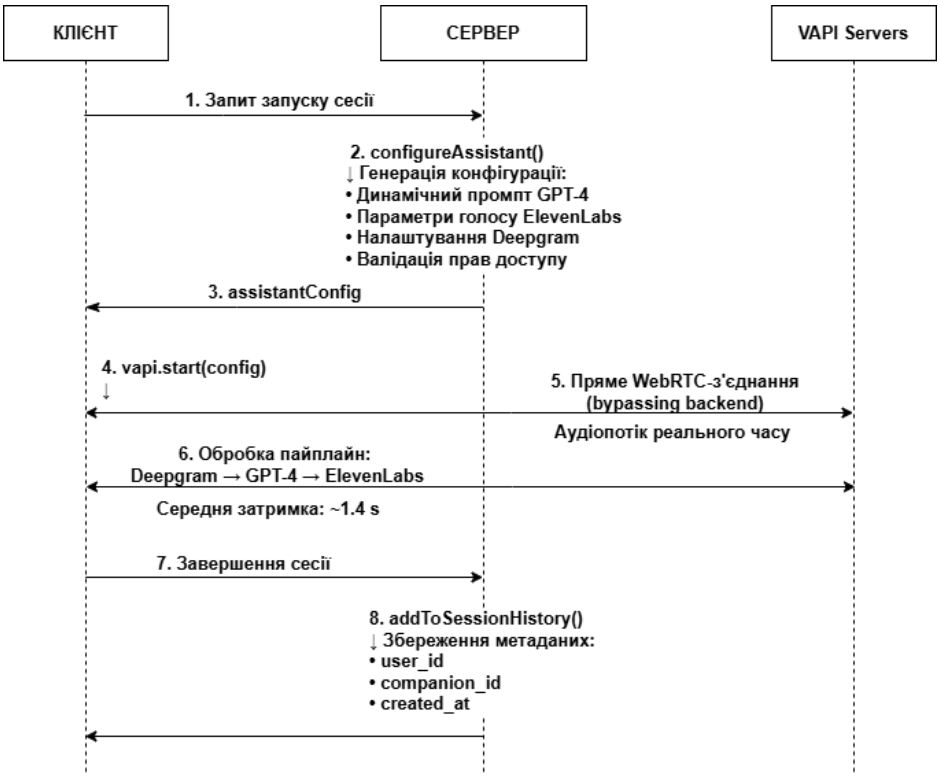

Рис. 2.3. Схема гібридної архітектури інтеграції Vapi

Після отримання конфігурації клієнт встановлює пряме WebRTC-з’єднання з Vapi для передачі аудіопотоку без проксіювання через власний backend. Браузер захоплює аудіо з мікрофона та передає його до Vapi, де воно послідовно транскрибується через Deepgram, обробляється GPT-4 з урахуванням історії діалогу та синтезується у голос через ElevenLabs, після чого повертається до браузера для відтворення. Повний цикл від завершення репліки користувача до початку відповіді асистента триває 1–2 секунди. Транскрипти обробляються через обробник події message, який фільтрує фінальні результати розпізнавання та додає їх до масиву messages для відображення в інтерфейсі.

Під час завершення сесії користувач викликає vapi.stop() для розриву WebRTC-з’єднання, що тригерить серверну функцію addToSessionHistory(), яка створює запис у таблиці session_history з метаданими сесії без збереження повного транскрипту для економії простору.

Розроблений метод гібридної архітектури забезпечує баланс між централізованим контролем персоналізації через серверну генерацію конфігурації, безпекою через приховування AI-промптів від клієнта, продуктивністю завдяки прямому WebRTC-з’єднанню та операційною стійкістю через мінімальне навантаження на власну інфраструктуру.

2.5 Висновки до розділу

У другому розділі проведено проєктування архітектури SaaS-платформи для AI-голосового навчання з персоналізованими асистентами. Аналіз проблематики та порівняння існуючих рішень показали відсутність комплексних платформ, що одночасно забезпечують голосову взаємодію для різних навчальних дисциплін, персоналізацію асистентів та структуроване відстеження прогресу користувачів. На основі виявлених обмежень сформульовано повний набір функціональних і нефункціональних вимог до системи.

Розроблена архітектурна модель базується на монолітному підході з безсерверним виконанням, що забезпечує оптимальний баланс між простотою розробки, автоматичним масштабуванням та економічною ефективністю. Архітектура відповідає четвертому рівню зрілості SaaS з логічною ізоляцією даних на основі RLS, інтеграцію ключових зовнішніх сервісів, а також гібридний підхід до обробки голосових сесій, спрямований на мінімізацію затримок.

Обґрунтовано вибір технологічного стеку, спроєктовано структуру бази даних із трьома доменними таблицями та визначено механізми багаторівневого захисту інформації. Крім того, розроблено алгоритми автентифікації, управління підписками, контролю доступу та інтеграції з AI-сервісами, які формують технологічне підґрунтя для подальшої програмної реалізації та впровадження платформи.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ SAAS-ПЛАТФОРМИ

3.1 Інтеграція backend-компонентів

3.1.1 Інтеграція Clerk для автентифікації та управління підписками

Інтеграція сервісу Clerk у SaaS-платформу для AI-голосового навчання забезпечує повний цикл управління користувачами – від реєстрації й автентифікації до монетизації, контролю доступу та підтримки різних моделей підписок. Вибір Clerk зумовлений наявністю готових UI-компонентів, вбудованими механізмами безпеки, інтеграцією зі Stripe, а також можливістю централізовано керувати тарифами, лімітами та властивостями користувачів без необхідності реалізовувати складну інфраструктуру вручну.

Процес інтеграції розпочинається зі створення проєкту в Clerk Dashboard, після чого система генерує набір API-ключів для клієнтської та серверної частини. Ці ключі додаються до конфігураційного файлу середовища розробки, що забезпечує коректну ініціалізацію сервісу в Next.js-додатку. На цьому етапі також конфігуруються домени для локального та продакшн-середовища, що забезпечує коректну роботу перенаправлень та збереження cookies автентифікації. Далі у панелі керування активуються необхідні методи входу – автентифікація через email та пароль, OAuth-провайдери Google, а також підтвердження електронної пошти.

У програмному коді Clerk інтегрується через офіційний SDK, який підключається до проєкту за допомогою пакета @clerk/nextjs. Базова конфігурація передбачає обгортання всього додатку у компонент ClerkProvider, що надає контекст автентифікації всім сторінкам і дає змогу використовувати інформацію про користувача через хуки useUser() та useAuth(). На рівні маршрутизації middleware визначає захищені сторінки та автоматично перенаправляє неавторизованих користувачів. Для відображення інтерфейсу автентифікації використовуються готові компоненти SignIn і SignUp, інтегровані у відповідні сторінки додатку, а компонент UserButton забезпечує взаємодію із профілем користувача.

Монетизація реалізується через модуль «Billing» у Dashboard, де Clerk підключається до Stripe і дозволяє створювати тарифні плани без необхідності будувати власну платіжну логіку. У межах платформи визначено три плани: Free (3 Companions), Standard (10 Companions) і Unlimited, які зображені на рисунку 3.1. Вони відрізняються обмеженнями на кількість створених AI-асистентів, що задаються через механізм features. Ці обмеження вказуються не в коді, а у Dashboard, що забезпечує можливість швидко змінювати параметри планів без повторного розгортання додатку.

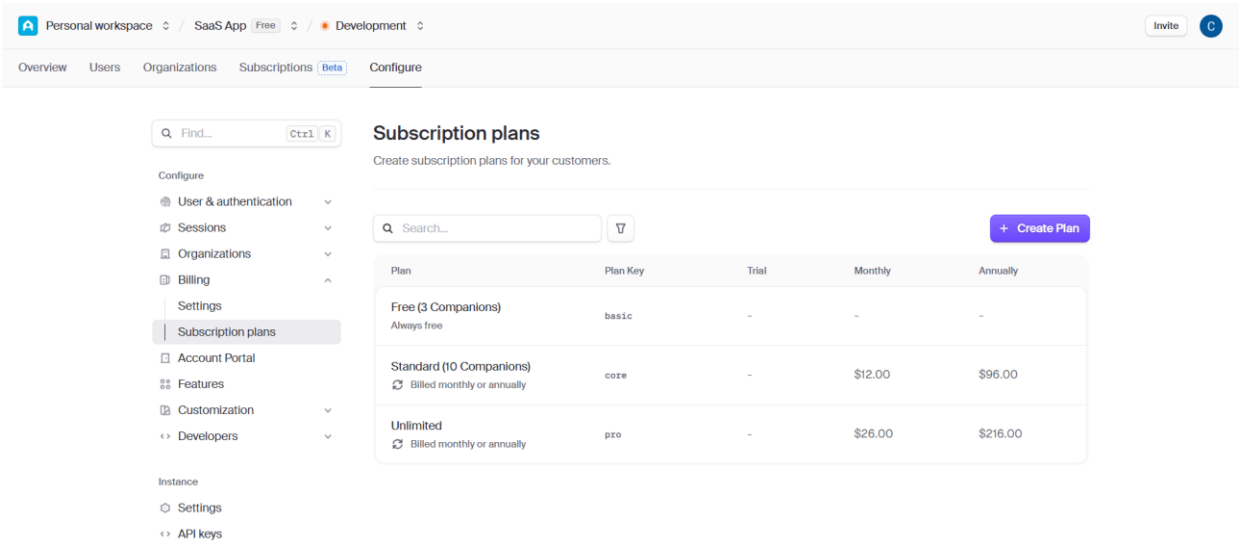

Рис. 3.1. Основні тарифні плани

Для відображення тарифів використовується компонент PricingTable, який автоматично зчитує налаштовані у Dashboard плани, відображає їх у візуальному форматі з назвами, цінами, переліком функцій, кнопками оформлення підписки. Компонент автоматично визначає поточний план користувача з метаданих та відображає відповідний статус підписки з кнопками апгрейду або даунгрейду.

У результаті інтеграція Clerk забезпечує надійну, безпечну та масштабовану систему управління автентифікацією, профілями та підписками. Завдяки цьому платформа може підтримувати складні моделі доступу, зберігаючи при цьому простоту реалізації, мінімізуючи внутрішню логіку та зменшуючи ризики, пов’язані з обробкою персональних даних і платежів.

3.1.2 Налаштування Supabase для зберігання даних

Інтеграція Supabase у SaaS-платформу навчання забезпечує надійне та структуроване зберігання даних, необхідне для коректної роботи всієї системи. Використання цього сервісу дає змогу поєднати керовану базу даних PostgreSQL, зручний API-доступ, вбудовані механізми безпеки та підтримку реального часу в єдиному інструменті.

Процес налаштування бази даних починається зі створення окремого проєкту у Supabase Dashboard, де ініціалізується інстанс PostgreSQL з автоматично згенерованими ключами доступу та URL-адресою API. Публічний ключ використовується на клієнтській стороні виключно під активними політиками безпеки, тоді як секретний сервісний ключ залишається у середовищі серверного виконання. Після створення проєкту відбувається налаштування таблиць: companions, session_history та bookmarks (рис. 3.2), які становлять основу даних платформи.

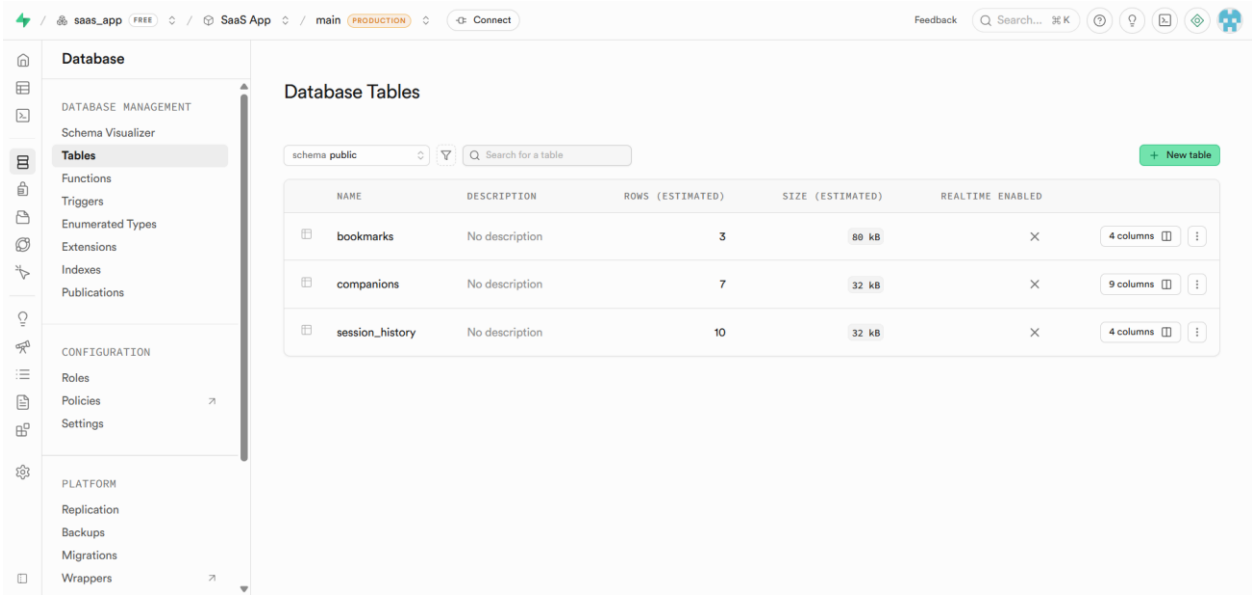

Рис. 3.2. Створені таблиці у Supabase

Після створення таблиць налаштовується фундаментальний механізм безпеки – Row Level Security. Саме RLS гарантує, що кожен користувач може оперувати лише власними даними, незалежно від того, як саме здійснюється запит: через клієнтський JavaScript-SDK, серверний код або навіть прямі HTTP-виклики.

Для таблиці companions формуються політики SELECT, INSERT, UPDATE та DELETE, які дозволяють доступ лише якщо значення поля author збігається з ідентифікатором автентифікованого користувача. Стандартна функція auth.uid() Supabase замінюється на власну функцію, тому що платформа використовує Clerk замість Supabase Auth. Для цього створюється кастомна функція get_user_id(), яка отримує ідентифікатор користувача з JWT-токену Clerk, переданого через заголовок запиту. Після цього усі політики переписуються так, щоб використовувати get_user_id() замість auth.uid().

Аналогічний підхід використовується для таблиць session_history та bookmarks. Політики для session_history дозволяють виконувати SELECT та INSERT лише для власних записів, тоді як UPDATE та DELETE забороняються, оскільки історія подій не повинна змінюватися після створення. У таблиці bookmarks політики SELECT, INSERT та DELETE також використовують user_id = get_user_id() як основну умову. Таким чином, RLS повністю ізолює дані між користувачами та унеможливлює доступ до чужих записів навіть у разі прямого виклику REST API.

Інтеграція Supabase у Next.js-додаток виконується через офіційний SDK. Після встановлення пакета @supabase/supabase-js створюється окремий модуль lib/supabase.ts, який інкапсулює логіку ініціалізації клієнта. У цьому модулі визначається фабрична функція createSupabaseClient, яка передбачає автоматичне отримання JWT-токену Clerk через callback accessToken(). Кожен запит до Supabase супроводжується передачею токену у заголовках, що забезпечує коректну роботу RLS-політик на сервері й унеможливлює доступ до даних без автентифікації. Додатково застосування TypeScript дозволяє уникнути значної кількості помилок завдяки типобезпечним запитам.

Робота з даними у проєкті відбувається через стандартні методи Supabase SDK: .from(), .select(), .insert(), .update() та .delete(). Приклади включають створення асистента, отримання списку власних записів, отримання історії сесій з вкладеними асистентами через JOIN-подібний синтаксис та фільтрацію даних на сервері. Обробка помилок виконується через перевірку поля error, де особливу увагу приділяють помилкам RLS та дублюванню унікальних значень.

У результаті проведеного налаштування Supabase забезпечує повністю безпечне, масштабоване та типобезпечне середовище зберігання даних, яке інтегрується з автентифікацією Clerk, підтримує мультитенантність за рахунок механізмів Row-Level Security та гарантує високу продуктивність завдяки оптимізованій схемі індексів.

3.1.3 Інтеграція Vapi для AI-голосових діалогів

Інтеграція платформи оркестрації голосових діалогів Vapi є ключовим технічним рішенням, що забезпечує проведення навчальних сесій через природну голосову взаємодію. Vapi обрано завдяки здатності поєднувати окремі AI-технології (розпізнавання мовлення Deepgram, обробку природної мови GPT-4, синтез мовлення ElevenLabs) в єдиний оптимізований пайплайн реального часу. Платформа пропонує стандартизований та простий у використанні API, що дозволяє реалізувати голосові діалоги з мінімальним навантаженням на бекенд і без необхідності управляти складною інфраструктурою WebRTC та аудіостримінгу.

Після створення облікового запису на Vapi та ініціалізації нового проєкту генеруються два ключі доступу: публічний токен, що використовується на клієнтській стороні для ініціалізації WebRTC-з'єднання у браузері, та приватний серверний ключ, який застосовується для створення, модифікації та управління AI-асистентами через бекенд.

У проєкт Next.js бібліотека Vapi SDK інтегрується через пакет @vapi-ai/web, а її ініціалізація виконується у глобальному модулі vapi.sdk.ts. Конфігурація AI-асистента формується серверною функцією configureAssistant(), що приймає параметри companion (голос, стиль, предметну область, тему) і повертає структуру CreateAssistantDTO. Конфігурація включає налаштування мовної моделі GPT-4, розпізнавання Deepgram Nova-3, синтезу ElevenLabs із параметрами стабільності та швидкості, а також системний промпт, що визначає поведінку асистента як репетитора. Промпт задає стиль комунікації (formal/casual), структуру пояснень і підтримку діалогу. Змінні {{topic}}, {{subject}}, {{style}} динамічно підставляються під час запуску сесії.

На клієнтській стороні логіка реалізована у React-компоненті CompanionComponent. Під час монтування ініціалізуються слухачі подій (call-start, call-end, speech-start, speech-end, message, error), що дають змогу оновлювати інтерфейс, керувати історією повідомлень і відображати транскрипти. Обробник message відповідає за прийом фінальних транскриптів, які додаються до масиву повідомлень.

Запуск сесії включає генерацію конфігурації асистента, формування параметрів підстановки та виклик vapi.start(), що встановлює WebRTC-канал між браузером і серверами Vapi. Обмін аудіоданими відбувається без участі бекенду платформи, що є критичним для низької латентності – у середньому 1-2 секунди. Браузер передає аудіо до Deepgram для розпізнавання, текст обробляється GPT-4, відповідь синтезується ElevenLabs і повертається до браузера.

Транскрипти оновлюються в реальному часі через фільтрацію фінальних результатів розпізнавання. Кожна репліка структурується з роллю (користувач/асистент) і текстом, додається до масиву messages у зворотному хронологічному порядку. Контейнер транскрипту підтримує прокручування, адаптується до мобільних пристроїв і використовує градієнтний ефект для індикації додаткового контенту.

Для користувача реалізовано можливість тимчасово вимикати мікрофон без зупинки сесії, що є корисною функцією під час навчального процесу. Завершення сесії виконується викликом vapi.stop(), який зупиняє аудіопотік, розриває WebRTC-з’єднання та тригерить подію call-end. Після завершення сесії інформація про неї зберігається у базі даних для формування навчальної статистики.

Основними перевагами інтеграції Vapi у SaaS-систему є:

- підтримка стійкої голосової взаємодії з низькою затримкою завдяки WebRTC;
- гнучка конфігурація AI-асистента з можливістю персоналізації голосу, стилю та поведінки;
- використання перевірених AI-сервісів (Deepgram, GPT-4, ElevenLabs) у єдиному технологічному ланцюжку;
- мінімальне навантаження на бекенд системи;

- коректне управління життєвим циклом сесії та станами клієнтського інтерфейсу;
- структурована система транскриптів і збереження історії взаємодій.

Таким чином, інтеграція Vapi забезпечує фундаментальну функціональність голосового навчання, дозволяючи системі працювати у режимі реального часу, підтримувати високу якість діалогу, формувати індивідуальний стиль взаємодії з користувачем і значно підвищувати ефективність навчального процесу через природний голосовий інтерфейс.

3.2 Реалізація клієнтської частини

3.2.1 Дизайн та адаптивність інтерфейсу

Дизайн користувацького інтерфейсу SaaS-платформи для AI-голосового навчання розроблений у мінімалістичному стилі з акцентом на інтуїтивність, чисту візуальну ієрархію та зручність під час взаємодії. Основою виступає темна кольорова схема з фіолетовими акцентами, що підкреслює технологічний характер системи та зменшує втому очей під час тривалих сесій.

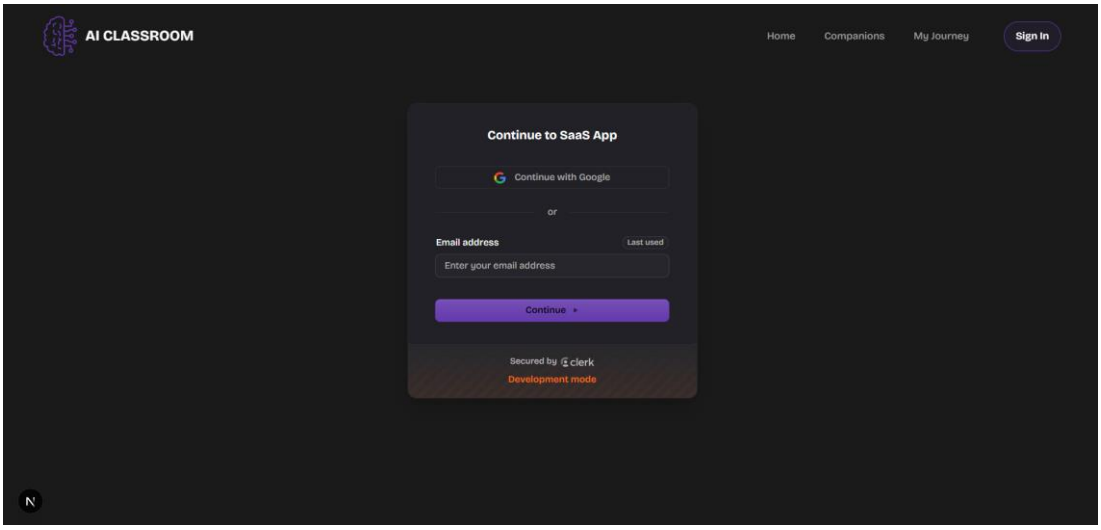

Рис. 3.3. Сторінка входу до системи з формою автентифікації

Сторінка входу до системи (рис. 3.3) представлена як центрована форма автентифікації на темному фоні, оформлена у мінімалістичному стилі, що концентрує увагу користувача виключно на процесі входу. Компонент SignIn від Clerk забезпечує

автоматичне формування адаптивної форми з основними полями (електронна пошта та пароль), а також підтримку альтернативних способів автентифікації через Google.

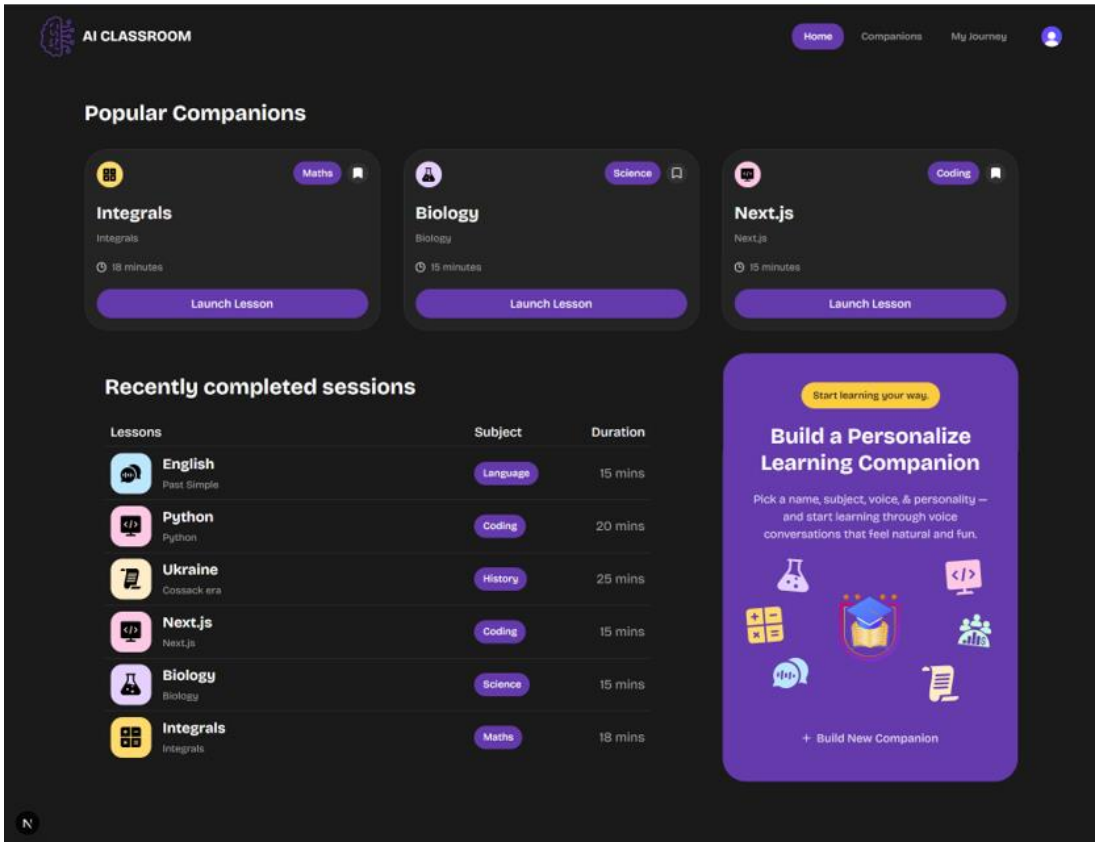

Рис. 3.4. Головна сторінка платформи з популярними асистентами

Головна сторінка платформи (рис. 3.4) представляє користувачу огляд доступних AI-асистентів та швидкий доступ до ключових функцій. Верхня панель навігації містить логотип платформи зліва, горизонтальне меню з розділами «Home», «Companions», «My Journey» та кнопку входу або аватар користувача справа. Основний контент сторінки включає відображення трьох популярних асистентів у вигляді карток, секцію «Recently completed sessions» зі списком нещодавніх навчальних сесій, а також блок для створення персоналізованого асистента з кнопкою «Build New Companion».

Сторінка бібліотеки асистентів (рис. 3.5) надає повний перелік доступних AI-асистентів з можливостями пошуку та фільтрації. Картки асистентів містять кольорову іконку предмету, бейдж з назвою предмету, назву асистента, короткий опис теми, тривалість сесії та кнопку «Launch Lesson» на всю ширину картки.

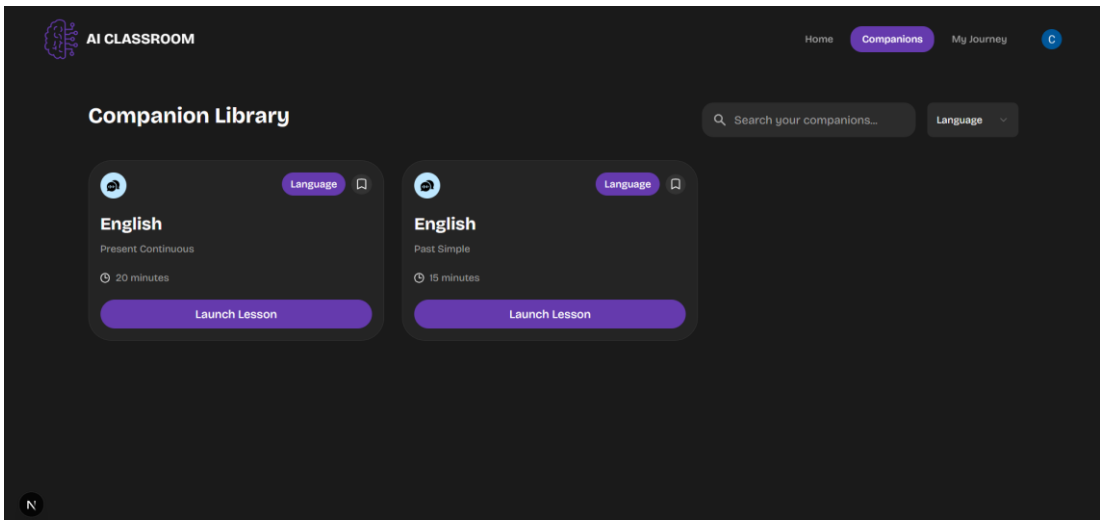

Рис. 3.5. Бібліотека AI-асистентів з пошуком та фільтрацією

Сторінка створення нового асистента (рис. 3.6) представляє форму з послідовними полями для налаштування параметрів. Форма організована вертикально з чіткими підписами полів: текстове поле для назви асистента, випадаючий список вибору предмету, текстова область для опису теми, радіокнопки вибору типу голосу та стилю комунікації, числове поле тривалості сесії у хвилинах. Кнопка «Build Your Companion» завершує форму.

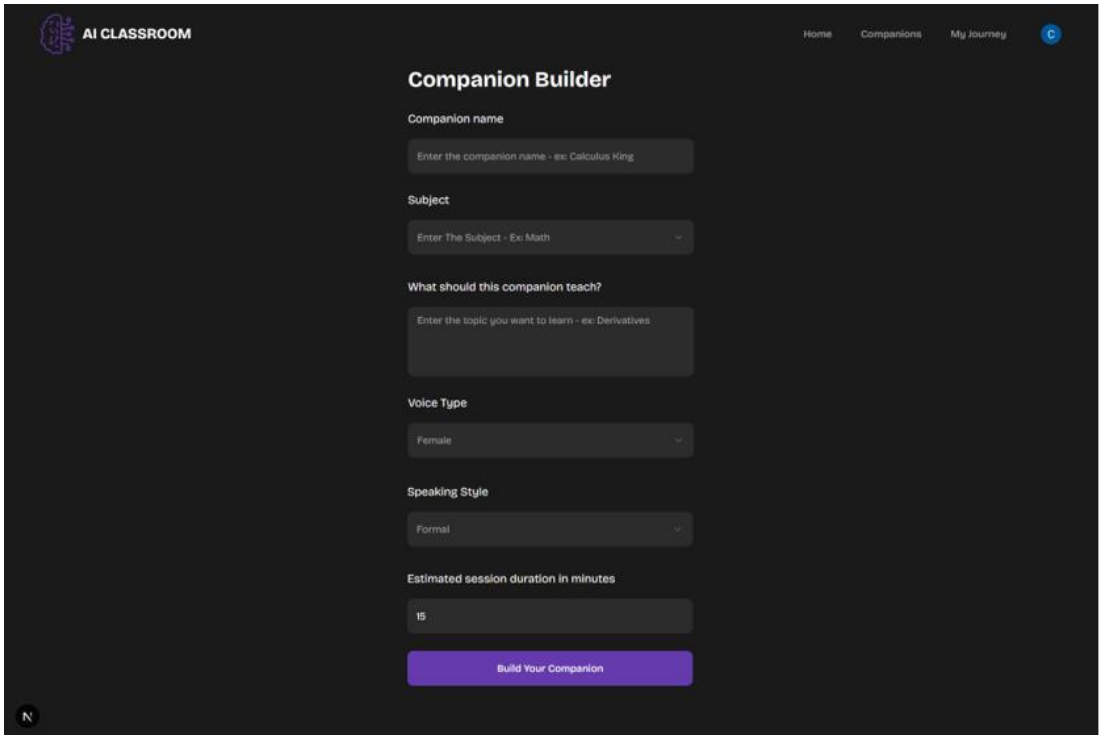

Рис. 3.6. Форма створення персоналізованого AI-асистента

Сторінка активної голосової сесії (рис. 3.7) оптимізована під живу взаємодію. Верхня інформаційна панель відображає назву асистента, предмет та тему у компактному форматі. Основний контент розділений на дві секції: ліва містить іконку асистента у неактивному стані або анімовані звукові хвилі під час його мовлення; права секція відображає аватар користувача, кнопку керування мікрофоном «Turn on/off microphone», кнопку завершення сесії «End Session» або кнопку «Start Session» перед початком. Нижня частина екрану відведена під транскрипт діалогу з автоматичним скролом.

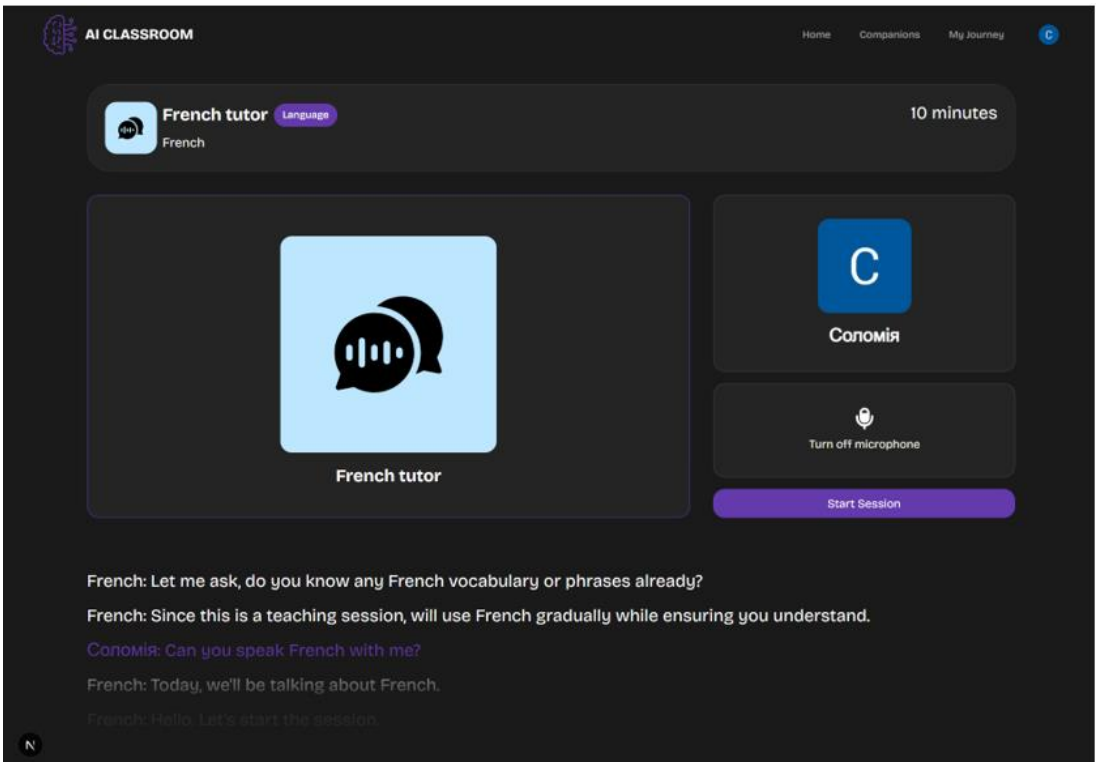

Рис. 3.7. Інтерфейс активної голосової навчальної сесії

Сторінка профілю користувача (рис. 3.8) агрегує персональну інформацію, статистику та історію навчання. Верхня секція містить аватар користувача, ім'я та електронну пошту, дві статистичні картки з кількістю завершених сесій та створених асистентів. Нижче розташовані три акордеони з секціями: зі списком збережених асистентів, з таблицею останніх навчальних сесій та зі списком створених користувачем асистентів. Кожна секція розгортається при натисканні на заголовок, відображаючи детальну таблицю з іконками предметів, назвами асистентів та темами.

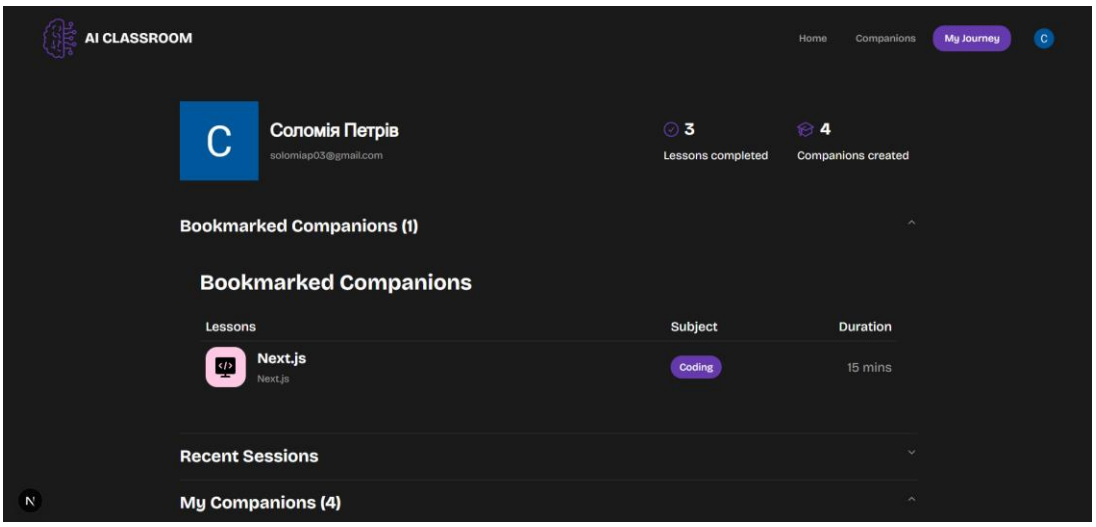

Рис. 3.8. Профіль користувача з історією навчання та статистикою

Сторінка тарифних планів (рис. 3.9) представляє доступні підписки через компонент PricingTable від Clerk з автоматичною інтеграцією з системою платежів. Кожна картка містить назву плану великим шрифтом, вартість жирним шрифтом, список включених функцій з іконками галочок, кнопку оформлення підписки.

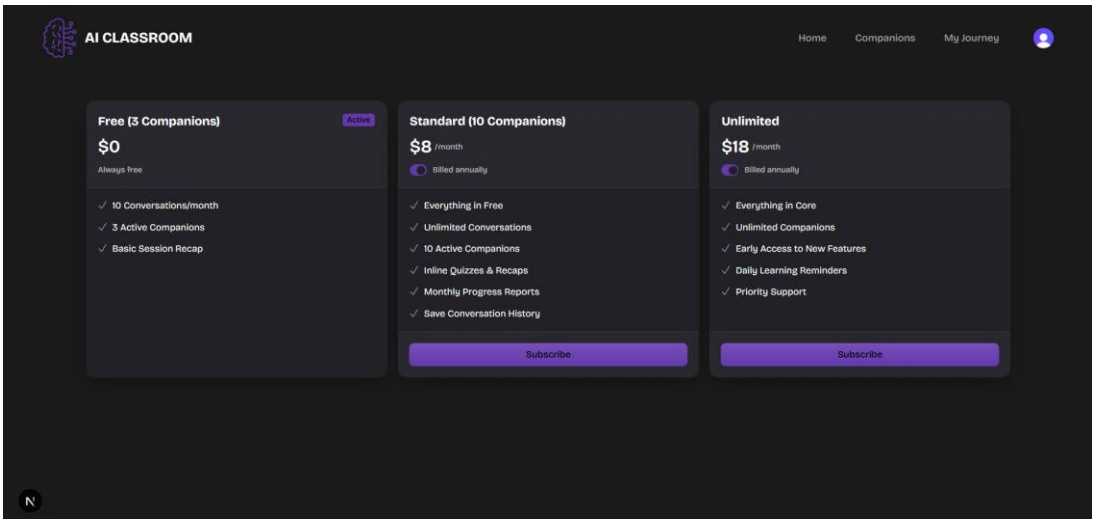

Рис. 3.9. Сторінка вибору тарифного плану підписки

Адаптивність інтерфейсу платформи ґрунтується на застосуванні mobile-first підходу та системи контрольних точок Tailwind CSS, що дозволяє масштабувати всі елементи відповідно до типу пристрою. Це передбачає трансформацію компонування від багатоколонкових структур на настільних пристроях до вертикальних стеків на

мобільних екранах. Графічні елементи, текстові блоки та інтерактивні компоненти (кнопки, форми, картки, таблиці) автоматично змінюють розміри та відступи, що гарантує збереження читабельності, комфортну взаємодію та відсутність горизонтальної прокрутки.

Адаптивна поведінка ключових екранів: головної сторінки, бібліотеки асистентів, форм створення, активної голосової сесії та профілю користувача – передбачає пропорційне масштабування візуальних елементів, приховування другорядних колонок у таблицях, перебудову порядку блоків та розширення інтерактивних елементів на всю ширину мобільного контейнера. Завдяки цьому респонсив-дизайн забезпечує узгоджений вигляд і функціональність на всіх типах пристроїв та підтримує цілісність візуальної системи, закладеної в дизайн-макетах.

Такий інтерфейс забезпечує консистентний візуальний досвід у всіх розділах платформи, інтуїтивну навігацію через чітку ієрархію інформації та стандартні патерни взаємодії, фокус на ключовому функціоналі через мінімалістичний дизайн без зайвих елементів та доступність для користувачів з різним рівнем технічної підготовки через зрозумілі логічні потоки дій.

3.2.2 Імплементація компонентів

Імплементація користувацького інтерфейсу SaaS-платформи базується на компонентному підході React з використанням Tailwind CSS для стилізації та бібліотеки shadcn/ui для готових елементів. Така архітектура забезпечує модульність, повторне використання коду та спрощує підтримку завдяки чіткому розділенню відповідальності між компонентами. Кожен компонент є самодостатньою одиницею з власною логікою стану, обробниками подій та стилізацією, що дозволяє незалежно розробляти, тестувати та оптимізувати окремі частини інтерфейсу.

Навігаційна панель реалізує верхню панель з логотипом, меню розділів та елементами автентифікації. Компоненти SignedIn та SignedOut від Clerk забезпечують умовний рендеринг для авторизованих та неавторизованих користувачів, а NavItems виділяє активний розділ. Використання напівпрозорого фону з розмиттям створює сучасний ефект накладення на контент.

Картки асистентів інкапсулюють відображення AI-асистентів у бібліотеці та на головній сторінці. Вони містять інформацію про назву, тему, предмет, тривалість сесії та статус закладки. Кольорові іконки предметів генеруються динамічно, а кнопки закладок синхронізуються з сервером через функції addBookmark та removeBookmark, що забезпечує автоматичне оновлення інтерфейсу.

Форма створення асистента реалізована через react-hook-form із інтеграцією Zod для валідації. Кожне поле пов'язане з контролером форми та забезпечує відображення повідомлень про помилки валідації. Обробник onSubmit передає дані на сервер і перенаправляє користувача на сторінку створеного асистента або повертає на головну у разі помилки.

Активна голосова сесія управляє станом сесії та взаємодією з Vapi. Локальні стани включають фази сесії, статус мовлення асистента, стан мікрофону та масив повідомлень. Використання useEffect забезпечує підписку на події Vapi та контроль анімації через Lottie, а обробники handleCall та handleDisconnect керують запуском і завершенням сесії.

Списки та пошук реалізують табличне та фільтраційне відображення асистентів. Пошук використовує дебаунсовану фільтрацію із синхронізацією стану з URL, а фільтри предметів оновлюють параметри URL для коректного відображення результатів. Блоки заклику до дії створюють промо-секції для залучення користувачів до створення асистентів. Вони реалізовані через градієнтний фон, чітку типографіку, кольорові бейджі та інтерактивні кнопки, що відповідають загальній дизайн-системі.

Таким чином, імплементація компонентів забезпечує модульність, підтримуваність, продуктивність та типобезпечність через застосування React, TypeScript та бібліотек для UI, що дозволяє ефективно розвивати інтерфейс платформи та підтримувати консистентний користувацький досвід.

3.3 Тестування системи

Тестування SaaS-платформи для AI-голосового навчання проводилось на трьох рівнях: функціональне тестування для перевірки коректності роботи основних

можливостей системи, тестування продуктивності для оцінки швидкодії та затримок критичних операцій, тестування безпеки для виявлення вразливостей ізоляції даних та механізмів контролю доступу. Комбінація ручного тестування через користувацький інтерфейс та автоматизованих інструментів моніторингу дозволила виявити та усунути проблеми на ранніх етапах розробки, забезпечуючи стабільність та надійність платформи перед розгортанням у продакшн-середовище.

Функціональне тестування проводилося для перевірки коректності роботи всіх ключових функцій системи відповідно до функціональних вимог, визначених у розділі 3. Кожен сценарій перевірявся на коректність обробки валідних даних, адекватність обробки помилкових вхідних даних, відповідність поведінки інтерфейсу очікуванням користувача, збереження стану між переходами сторінок.

Тестування охоплювало кілька основних функцій системи:

1. Автентифікація та управління користувачами – перевірено створення облікових записів через електронну пошту та OAuth, автоматичне оновлення токенів, перенаправлення при доступі до захищених маршрутів та завершення сесій.
2. Створення та управління AI-асистентами – перевірено валідацію форм, динамічне перенаправлення після успішного створення, контроль лімітів підписки та відображення списку асистентів у розділі користувача.
3. Голосові навчальні сесії – тестування WebRTC-з’єднання з Vapi, запуск сесій, розпізнавання мови користувача, генерація відповідей AI-асистента, керування мікрофоном та завершення сесії з автоматичним збереженням у базі.
4. Система закладок – перевірено створення, видалення та синхронізацію закладок між пристроями, а також коректне відображення у профілі користувача.
5. Пошук та фільтрація асистентів – перевірено динамічне оновлення URL та відображення результатів за параметрами topic та subject, комбіноване застосування пошуку і фільтрів, очищення параметрів.
6. Управління підписками – перевірено інтеграцію з сервісом Clerk, оформлення та оновлення тарифних планів, контроль доступу після зміни підписки.

Результати тестування задокументовані у таблиці А.1 (додаток А), що містить унікальні ідентифікатори тест-кейсів, модуль, тестові сценарії, вхідні дані, очікувані

та фактичні результати, а також статус виконання. Всі критичні функціональні можливості системи пройшли перевірку успішно, ідентифіковані незначні UX-проблеми не впливали на працездатність. Таким чином, проведене функціональне тестування підтвердило відповідність платформи функціональним вимогам та готовність до подальших етапів тестування.

Тестування продуктивності SaaS-платформи для AI-голосового навчання проводилося з метою оцінки швидкодії, стабільності роботи та відповідності встановленим нефункціональним вимогам, визначеним у підрозділі 3.2.1. Особливу увагу приділено латентності голосової взаємодії, швидкості завантаження інтерфейсу та ефективності обробки даних, оскільки ці параметри визначають якість користувацького досвіду в системах реального часу.

Аудит Lighthouse застосовувався до ключових сторінок платформи та аналізує їх за п’ятьма категоріями: Performance, Accessibility, Best Practices, SEO-оптимізація та прогресивний веб-додаток (PWA). Результати аудиту задокументовані у таблиці А.2 (додаток А), вони показали високі показники продуктивності у всіх категоріях. Найнижча оцінка на сторінці голосової сесії пояснюється завантаженням додаткових бібліотек Vapi SDK та Lottie, що є необхідним для функціоналу реалтайм-взаємодії. Показники доступності, що наближаються до 100 балів, підтверджують відповідність вимогам WCAG 2.1, а максимальні значення Best Practices і SEO свідчать про коректну імплементацію сучасних стандартів веб-розробки.

Латентність голосової взаємодії є критичною метрикою для платформи AI-голосового навчання, оскільки затримки понад 2-3 секунди руйнують відчуття природного діалогу. Вимірювання проводилося через логування часових міток у компоненті CompanionComponent з використанням performance.now() для високоточного вимірювання часу. Виміряно латентність від завершення мовлення користувача (подія speech-end) до початку відтворення відповіді асистента (подія speech-start). Середня латентність відповіді становить 1.4 секунди, що відповідає нефункціональним вимогам.

Тестування безпеки фокусувалось на перевірці ізоляції даних між користувачами через Row Level Security та контролю доступу до функцій на основі

підписки. Спроба доступу користувача B до асистента користувача A через прямий перехід за URL не повертала жодних даних, оскільки RLS-політика фільтрувала записи за author = auth.uid(). Аналогічно блокувались спроби створити закладку для асистента іншого користувача, тому що значення user_id встановлювалося автоматично на сервері та не могло бути підмінене.

Перевірка обходу тарифних обмежень шляхом модифікації клієнтського запиту через DevTools також виявилась безрезультатною: серверна функція createCompanion() виконувала перевірку newCompanionPermissions() незалежно від клієнта та блокувала операцію. Тест SQL-ін'єкції з введенням шкідливого рядка також не призвів до жодних негативних наслідків завдяки параметризованим запитам у Supabase та валідації вхідних даних через Zod-схеми.

Результати тестування підтверджують готовність платформи до розгортання у продакшн-середовище з високим рівнем функціональної коректності (100% пройдених тест-кейсів), прийнятною продуктивністю (Lighthouse Performance 92/100, латентність голосу 1.4 с), та надійною безпекою та ізоляцією даних (0 успішних атак на доступ до чужих даних).

3.4 Розгортання та моніторинг системи

3.4.1 Конвеєр розгортання через Vercel

Розгортання SaaS-платформи для AI-голосового навчання здійснюється за допомогою платформи Vercel, яка забезпечує автоматизований конвеєр безперервної інтеграції та доставки (CI/CD). Такий вибір зумовлений нативною підтримкою фреймворку Next.js, можливістю автоматичного масштабування безсерверних функцій, використанням глобальної мережі доставки контенту та наявністю механізмів попереднього перегляду. Система розгортання побудована на Git-орієнтованому підході, за якого кожен коміт у репозиторій GitHub автоматично ініціює процес збірки й розгортання, що усуває потребу в ручних операціях.

Інтеграція з GitHub виконується під час початкового налаштування, після чого Vercel отримує доступ до репозиторію та ідентифікує застосунок як Next.js-проєкт.

Основною продакшн-гілкою визначається main або master, з якої здійснюється автоматичне розгортання у виробниче середовище. Процес CI/CD активується через webhook-сповіщення GitHub після виконання push чи завершення pull request. У відповідь на це Vercel клонує репозиторій, встановлює залежності, виконує команду npm run build, компілює код, оптимізує бандли та розміщує результат у глобальній Edge-мережі. Повний цикл розгортання зазвичай триває 2–3 хвилини.

Платформа реалізує механізм zero-downtime deployment, за якого нова версія активується лише після успішної перевірки її працездатності, тоді як попередня версія продовжує обслуговувати користувачів до моменту перемикання трафіку. У разі помилки збірки розгортання анулюється, а продакшн залишається незмінним. Логи збірки доступні для аналізу у Vercel Dashboard, а функціонал Instant Rollback дає змогу швидко повернутися до попередньої версії.

Середовища попереднього перегляду створюються для кожного pull request, надаючи ізольований простір для перевірки функціональних змін. Виони використовують той самий набір змінних та налаштувань, що і продакшн, забезпечуючи ідентичні умови виконання. Після завершення pull request відповідне середовище видаляється для оптимізації використання ресурсів.

3.4.2 Моніторинг та відстеження помилок через Sentry

Система моніторингу та відстеження помилок у SaaS-платформі реалізована шляхом інтеграції сервісу Sentry на всіх рівнях застосунку, що забезпечує своєчасне виявлення інцидентів у продакшн-середовищі, оперативну діагностику причин їх виникнення та підтримання стабільної роботи платформи. Sentry автоматично фіксує виключення, необроблені відхилення промісів, помилки React-компонентів на клієнті, серверні помилки у функціях Next.js, а зібрані дані агрегуються у звіти зі контекстом виконання та інформацією про пристрій і користувача.

Ініціалізація Sentry охоплює три середовища виконання коду:

- instrumentation-client.ts конфігурує клієнтський JavaScript, задаючи DSN для ідентифікації проєкту, частку зразків для трасування продуктивності, параметри запису користувацьких сесій для фіксації DOM-подій.

- sentry.server.config.ts відповідає за конфігурацію серверної частини Next.js, зокрема API-маршрутів та серверних компонентів.
- sentry.edge.config.ts забезпечує роботу Sentry у middleware та функціях, що виконуються на Edge Runtime.

Захоплення виключень здійснюється автоматично без необхідності ручного додавання try–catch конструкцій. На клієнті Sentry перехоплює події window.onerror, unhandledrejection, а також помилки React через механізм componentDidCatch. На сервері Sentry інтегрується у внутрішню інфраструктуру Next.js і перехоплює виключення у серверних компонентах, API-функціях та Node.js-процесі. Усі помилки автоматично збагачуються додатковим контекстом: URL-адресою, параметрами запиту, HTTP-заголовками, даними про автентифікованого користувача, а також послідовністю подій, що передували інциденту. Механізм breadcrumbs фіксує найважливіші користувацькі дії та системні події до моменту виникнення помилки: переходи між сторінками, кліки по елементах інтерфейсу, мережеві запити, зміни стану застосунку та консольні повідомлення.

Підсистема Performance Monitoring автоматично відстежує тривалість виконання ключових транзакцій як на сервері, так і на клієнті. Для серверних функцій створюються транзакції з окремими span-вимірюваннями запитів до бази даних, зовнішніх API або обчислювальних операцій. На клієнті фіксується час завантаження сторінок, рендерингу компонентів та взаємодій.

Функція Session Replay забезпечує можливість відтворення сесій користувачів, під час яких виникли помилки. Запис включає зміни DOM, кліки, переходи між сторінками та мережеві запити, що дає змогу дослівно відтворити користувацький сценарій, який призвів до інциденту. Чутлива інформація автоматично маскується.

Механізм оповіщення Sentry формує сповіщення у випадку критичних подій: появи нової помилки, різкого збільшення частоти існуючих помилок, збоїв у функціональності, що впливає на значну кількість користувачів, або інцидентів у критичних бізнес-процесах. Сповіщення надсилаються електронною поштою або інтегруються з інструментами керування інцидентами, такими як Slack чи PagerDuty, що забезпечує оперативну реакцію команди.

Така система моніторингу забезпечує проактивне виявлення проблем до отримання скарг від користувачів, швидку діагностику причин помилок через детальний контекст та breadcrumbs, контроль якості через метрики частоти помилок та продуктивності та підтримання високого рівня надійності сервісу для користувачів платформи.

3.5 Висновки до розділу

У третьому розділі представлено практичну реалізацію SaaS-платформи для AI-голосового навчання, що підтверджує працездатність запропонованих моделей, методів та алгоритмів. Інтеграція backend-компонентів реалізована через спеціалізовані сервіси: Clerk забезпечує автентифікацію та управління підписками з автоматичною синхронізацією метаданих, Supabase з Row Level Security гарантує ізоляцію даних між користувачами на рівні бази даних, Vapi забезпечує голосову взаємодію через гібридну архітектуру з прямим WebRTC-з'єднанням для мінімізації латентності. Користувацький інтерфейс реалізовано через React та Tailwind CSS з повним набором адаптивних екранів для всіх ключових функцій платформи.

Результати тестування підтверджують готовність платформи до розгортання у продакшн-середовище з високим рівнем функціональної коректності (100% пройдених тест-кейсів), прийнятною продуктивністю (Lighthouse Performance 92/100, латентність голосу 1.4с), та надійною безпекою та ізоляцією даних (0 успішних атак на доступ до чужих даних). Розгортання через Vercel забезпечує автоматизований CI/CD з zero-downtime deployment, а Sentry гарантує проактивний моніторинг помилок та продуктивності.

Таким чином, практична реалізація підтверджує працездатність запропонованої архітектурної моделі, ефективність методу ізоляції даних через RLS, коректність алгоритму управління підписками через метадані та готовність платформи до комерційного впровадження з можливістю обслуговування сотень користувачів.

ВИСНОВКИ

У магістерській роботі вирішено актуальне науково-практичне завдання дослідження та удосконалення моделей, методів та алгоритмів побудови SaaS-платформ, що забезпечують ефективне управління ресурсами, безпеку даних, контроль доступу та інтеграцію зовнішніх сервісів в умовах мультитенантної архітектури. Розроблені рішення орієнтовані на застосування у освітніх платформах з AI-технологіями, але мають універсальний характер та можуть використовуватись для широкого класу SaaS-систем.

Проведений аналіз теоретичних основ Software as a Service показав, що існуючі моделі та методи часто орієнтовані на вирішення окремих задач без комплексного врахування специфіки SaaS-архітектури. Дослідження моделей ізоляції даних (Database-Level, Schema-Level, Row-Level) виявило їх переваги та обмеження, а аналіз методів автентифікації, управління підписками та розподілу ресурсів встановив необхідність об’єднання підходів для забезпечення оптимального балансу між продуктивністю та безпекою.

На основі виконаного аналізу розроблено архітектурну модель SaaS-платформи четвертого рівня зрілості, яка базується на монолітному підході з безсерверним виконанням та логічною ізоляцією даних. Модель забезпечує автоматичне масштабування, оплату за фактичне використання ресурсів та підтримку мультитенантності без дублювання інфраструктури. Використано метод ізоляції даних на основі Row-Level Security з інтеграцією JWT-токенів, який гарантує повну ізоляцію даних між користувачами на рівні бази даних. Метод передбачає автоматичне застосування політик безпеки до кожного SQL-запиту на основі ідентифікатора користувача, вбудованого у токен автентифікації, що унеможливлює доступ до чужих даних навіть при компрометації додаткових рівнів захисту.

Проаналізовано алгоритм контролю доступу на основі метаданих підписки, який забезпечує диференційований доступ до функціональності залежно від тарифного плану користувача. Алгоритм включає синхронізацію метаданих підписки з сервісу управління платежами у основну базу даних, перевірку обмежень на

клієнтській стороні для покращення користувацького досвіду, додаткову валідацію на серверній стороні для запобігання несанкціонованого доступу та автоматичне оновлення прав доступу при зміні тарифного плану.

Запропоновано метод гібридної інтеграції зовнішніх сервісів для мінімізації латентності взаємодії в реальному часі. Метод передбачає встановлення прямих з'єднань між клієнтським застосунком та спеціалізованими зовнішніми сервісами з одночасним збереженням централізованого контролю через токени доступу з обмеженим часом дії. Це дозволяє уникнути додаткової затримки від проксіювання трафіку через основний сервер при збереженні контролю над авторизацією та обліком використання ресурсів.

Практична апробація розроблених моделей, методів та алгоритмів здійснена через реалізацію SaaS-платформи для AI-голосового навчання. Експериментальна верифікація підтвердила ефективність розроблених рішень: метод ізоляції даних забезпечив 100% ізоляцію між користувачами при 0 успішних спроб несанкціонованого доступу, алгоритм контролю доступу коректно обмежує функціональність згідно з тарифним планом, метод гібридної інтеграції досяг латентності 1.4 секунди для взаємодії в реальному часі. Платформа успішно пройшла функціональне тестування (100% тест-кейсів), продемонструвала високу продуктивність (Lighthouse Performance 92/100) та готовність до автоматичного масштабування під навантаженням.

Подальший розвиток дослідження може включати удосконалення алгоритмів автоматичного масштабування з використанням предиктивної аналітики навантаження, розширення методів забезпечення безпеки для роботи з регульованими даними та розробку моделей оптимізації витрат на хмарну інфраструктуру при збереженні заданого рівня якості обслуговування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Германович, Сніжана Сергіївна. "Застосування хмарних технологій для реалізації рішень Інтернету речей." (2021).
2. Нечипоренко, Н. С. "Інтелектуальний аналіз міграції даних з фізичних серверів до хмарних обчислювальних середовищ." (2025).
3. Younis, Rehmana, et al. "A comprehensive analysis of cloud service models: IaaS, PaaS, and SaaS in the context of emerging technologies and trend." 2024 international conference on electrical, communication and computer engineering. IEEE, 2024.
4. Смірнова, Тетяна, et al. "Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS." Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка» 4.24 (2024): 6-27.
5. Wu, Cheng-Han, and Pandu Dwi Luhur Pambudi. "Digital transformation in fintech: Choosing between application and Software as a Service (SaaS)." Asia Pacific Management Review 30.2 (2025): 100342.
6. Golding, Tod. Building Multi-tenant SaaS Architectures: Principles, Practices, and Patterns Using AWS. " O'Reilly Media, Inc.", 2024.
7. Aleem, Saiqa, et al. "SaaS application maturity assessment model." IEEE Access 12 (2024): 128305-128325.
8. Pappula, Kiran Kumar. "Architectural Evolution: Transitioning from Monoliths to Service-Oriented Systems." International Journal of Emerging Research in Engineering and Technology 3.4 (2022): 53-62.
9. Zangana, Hewa Majeed, Zina Bibo Sallow, and Marwan Omar. "Cloud architectures for distributed serverless computing: A review of event-driven and function-as-a-service paradigms." International Journal of Artificial Intelligence & Robotics (IJAIR) 6.2 (2024): 57-64.
10. Шевченко, Олена Миколаївна, Євгеній Ігорович Верезомський, and Ірина Георгіївна Верезомська. "Оцінка впливу цифрових платформ на зміни в моделях бізнесу та конкурентні стратегії підприємств." (2025).
11. Коваленко, В. В., et al. "Цифрові освітні технології: дайджест." (2025).

12. Humayun, Mamoona, et al. "Software-as-a-service security challenges and best practices: A multivocal literature review." Applied Sciences 12.8 (2022): 3953.
13. Yassin, Mohamed, et al. "Multi-tenant intrusion detection framework as a service for SaaS." IEEE Transactions on Services Computing 15.5 (2021): 2925-2938.
14. Dar, Chen, Moshik Hershcovitch, and Adam Morrison. "RLS Side Channels: Investigating Leakage of Row-Level Security Protected Data Through Query Execution Time." Proceedings of the ACM on Management of Data 1.1 (2023): 1-25.
15. Sethuraman, Swaminathan, Chiranjeevi Devi, and Chandan Gnana Murthy. "Policy-as-Code Row-Level Security: Compiling DPL Rules into Spark SQL Views." American Journal of Data Science and Artificial Intelligence Innovations 2 (2022): 673-705.
16. Al-Ammouri, A., et al. "Методи та засоби захисту інформації." Системи управління, навігації та зв'язку. Збірник наукових праць 1.75 (2024): 38-44.
17. Морозюк, А. А., and П. К. Ніколюк. "Шифрування даних." Комп'ютерні технології обробки даних (2022): 80-82.
18. Бганцов Є. (2023) Шифрування даних. Опис та структура алгоритму симетричного шифрування AES, Abstracts of VI International Scientific and Practical Conference «Methodical and practical methods of creating inventions», (October 24 – 27, 2023). Sofia, Bulgaria, pp. 222-225.
19. Мокрій, В. С., Д. О. В'юхін, and А. В. Власов. Криптографічні механізми захисту смарт-контрактів. Diss. ХНУРЕ, 2025.
20. Баришев, Ю. В., and В. С. Ланова. Аналіз геш-функцій для захисту цілісності чутливих даних на основі технології блокчейн. Diss. ВНТУ, 2024.
21. Радівілова, Тамара, et al. "Аналіз методів автентифікації для вебзастосунків та реалізація вебзастосунку з інтегрованою системою автентифікації." Сучасний стан наукових досліджень та технологій в промисловості (2024): 76-90.
22. Палєга, Р. В., Н. В. Карпенко, and В. В. Герасимов. "Особливості авторизації за допомогою JWT токенів." Information technologies and automation (2024): 208.
23. Li, Wanpeng, and Chris J. Mitchell. "User access privacy in OAuth 2.0 and OpenID connect." 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 2020.

24. Холоша, О. С. "Метод захисту веб-добатків на основі протоколів Oauth 2.0 і OpenID." (2021).
25. Сусідко, Владислав, Нікіта Панагода, and Тамара Лященко. "Сучасні протоколи автентифікації та авторизації: аналіз OAuth 2.0, OIDC та SAML." (2024).
26. Dashti, Salimeh, et al. "Automated Risk Assessment and What-if Analysis of OpenID Connect and OAuth 2.0 Deployments." IFIP Annual Conference on Data and Applications Security and Privacy. Cham: Springer International Publishing, 2021.
27. Колешня, Я. О. "Сучасні цифрові бізнес-моделі: сутність, огляд та особливості." Підприємництво та інновації 24 (2022): 87-91.
28. Hatch, Ralph. SaaS Architecture, Adoption and Monetization of SaaS Projects using Best Practice Service Strategy, Service Design, Service Transition, Service Operation and Continual Service Improvement Processes. Emereo Pty Ltd, 2008.
29. Hinterhuber, Andreas. "SaaS Pricing." Digital Pricing Strategy: Capturing Value from Digital Innovations (2023): 71.
30. Bankole, Folake Ajoke, and L. Tewogbade. "Optimizing subscription cost structures in technology enterprises using scalable, data-informed forecasting techniques." International Journal of Scientific Engineering Research and Science Education and Technology 11.6 (2024): 359-392.
31. Пазинін, Андрій. "Методи автоматичного масштабування в хмарних середовищах." Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка» 2.26 (2024): 445-459.
32. Alharthi, Saleha, et al. "Auto-scaling techniques in cloud computing: Issues and research directions." Sensors 24.17 (2024): 5551.
33. Rathinam, Anantha Raman, et al. "Advances and Predictions in Predictive Auto-Scaling and Maintenance Algorithms for Cloud Computing." 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS). IEEE, 2023.
34. Shafiq, Dalia Abdulkareem, N. Z. Jhanjhi, and Azween Abdullah. "Load balancing techniques in cloud computing environment: A review." Journal of king saud university-computer and information sciences 34.7 (2022): 3910-3933.

35. Хмельницький, Юрій Владиславович, and Марія Євгеніївна Карун. "Балансування навантаження додатків у «хмарному» середовищі." (2018).
36. Ayezabu, Amanuel Zewdie. "Supabase vs Firebase: Evaluation of performance and development of Progressive Web Apps." (2022).
37. Sumanth, N. Sai, and S. Vishnu Priya. "AI-Enhanced Learning Assistant Platform." 2024 International Conference on Inventive Computation Technologies (ICICT). IEEE, 2024.
38. Garai, Pooja, et al. "Voice AI-Intelligence Based Voice Assistant." 2025 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE). IEEE, 2025.
39. Adeshola, I., & Adepoju, A. P. (2023). The opportunities and challenges of ChatGPT in education. Interactive Learning Environments, 1-14
40. Novak, Anton, and Tetyana Semigina. "Інтеграція ChatGPT у навчальну онлайн платформу (Integration of ChatGPT Into an Online Educational Platform)."

ДОДАТКИ

Додаток А

Таблиці з результатами тестування

Таблиця А.1

Результати функціонального тестування основних сценаріїв

ID	Модуль	Тест-кейс	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TC-01	Автентифікація	Реєстрація нового користувача	Email, Password	Обліковий запис створено, лист підтвердження надіслано	Обліковий запис створено, лист отримано	Pass
TC-02	Автентифікація	Вхід через Google OAuth	Google account	Автоматичний вхід, перенаправлення на головну	Вхід успішний, перенаправлення коректне	Pass
TC-03	Автентифікація	Доступ без автентифікації	URL /companions	Перенаправлення на /sign-in з збереженням URL	Перенаправлено на /sign-in?redirect_url=/companions	Pass
TC-04	Асистенти	Створення асистента	Name, Subject, Topic	Асистент створено, перенаправлення на сторінку сесії	Асистент створено з ID, перенаправлено	Pass
TC-05	Асистенти	Створення без назви	Name: ""	Помилка валідації "Name is required"	Червоне повідомлення під полем назви	Pass
TC-06	Асистенти	Перевірка ліміту (Free)	4-й асистент	Блокування з повідомленням про ліміт	Повідомлення "Subscription required", кнопка Upgrade	Pass
TC-07	Голосова сесія	Запуск сесії	Start Session	З'єднання встановлено, статус Active	З'єднання встановлено за ~2с, статус змінено	Pass
TC-08	Голосова сесія	Розпізнавання мови	Сказати "Hello"	Текст з'являється у транскрипті	"Hello" відображено як репліка user	Pass
TC-09	Голосова сесія	Відповідь AI	Запит "What is calculus?"	AI відповідає голосом та текстом	AI відповів за ~1.5с, текст у транскрипті	Pass
TC-10	Голосова сесія	Керування мікрофоном	Натиснути кнопку	Аудіо вимкнено, іконка змінена	Мікрофон вимкнено, іконка mic-off	Pass
TC-11	Голосова сесія	Завершення сесії	End Session	З'єднання розірвано, запис збережено	Сесія завершена, запис у session_history	Pass
TC-12	Закладки	Додавання закладки	Натиснути іконку	Іконка заповнюється, закладка збережена	Іконка змінена, запис у bookmarks	Pass
TC-13	Закладки	Видалення закладки	Натиснути заповнену іконку	Іконка порожня, закладка видалена	Іконка змінена, запис видалено	Pass
TC-14	Пошук	Пошук за темою	Ввести "derivatives"	Відфільтровано асистентів з темою	Показано 2 асистенти	Pass
TC-15	Фільтрація	Фільтр за предметом	Вибрати "maths"	Показано лише математичні асистенти	Відфільтровано 5 асистентів	Pass
TC-16	Підписка	Оформлення підписки	Вибрати Pro plan	Перенаправлення на Stripe Checkout	Перенаправлено на checkout.stripe.com	Pass
TC-17	Підписка	Перевірка доступу після підписки	Створити 11-го асистента	Асистент створено без обмежень	Асистент створено успішно	Pass
TC-18	Історія	Відображення історії	My Journey	Показано завершені сесії	Таблиця з 3 сесіями, дати коректні	Pass
TC-19	Профіль	Статистика	Переглянути статистику	Коректні числа сесій та асистентів	Lessons: 3, Companions: 5	Pass

Продовження додатку А

Таблиця А.2

Результати аудиту Lighthouse для основних сторінок

Сторінка	**Performance**	**Accessibility**	**Best Practices**	**SEO**	**Примітки**
Головна (/)	94/100	98/100	100/100	100/100	First Contentful Paint: 1.2s
Бібліотека (/companions)	92/100	98/100	100/100	100/100	Largest Contentful Paint: 1.8s
Створення (/companions/new)	95/100	100/100	100/100	91/100	Time to Interactive: 1.5s
Голосова сесія (/companions/[id])	89/100	95/100	100/100	100/100	Total Blocking Time: 120ms
Профіль (/my-journey)	93/100	98/100	100/100	100/100	Speed Index: 1.6s

Додаток Б

Реалізація голосової взаємодії через Vapi AI

```
import { clsx, type ClassValue } from "clsx";
import { twMerge } from "tailwind-merge";
import { subjectsColors, voices } from "@/constants";
import { CreateAssistantDTO } from "@vapi-ai/web/dist/api";
export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}
export const getSubjectColor = (subject: string) => {
  return subjectsColors[subject as keyof typeof subjectsColors];
};
export const configureAssistant = (voice: string, style: string) => {
  const voiceId = voices[voice as keyof typeof voices][
          style as keyof (typeof voices)[keyof typeof voices]
          ] || "sarah";
  const vapiAssistant: CreateAssistantDTO = {
    name: "Companion",
    firstMessage:
        "Hello, let's start the session. Today we'll be talking about {{topic}}.",
    transcriber: {
      provider: "deepgram",
      model: "nova-3",
      language: "en",
    },
    voice: {
      provider: "11labs",
      voiceId: voiceId,
      stability: 0.4,
      similarityBoost: 0.8,
      speed: 1,
      style: 0.5,
      useSpeakerBoost: true,
    },
    model: {
provider: "openai",
      model: "gpt-4",
      messages: [
        {
          role: "system",
          content: `You are a highly knowledgeable tutor teaching a real-time voice
session with a student. Your goal is to teach the student about the topic and subject.
                    Tutor Guidelines:
                    Stick to the given topic - {{ topic }} and subject - {{ subject }}
and teach the student about it.
                    Keep the conversation flowing smoothly while maintaining control.
                    From time to time make sure that the student is following you and
understands you.
                    Break down the topic into smaller parts and teach the student one
part at a time.
                    Keep your style of conversation {{ style }}.
                    Keep your responses short, like in a real voice conversation.
                   Do not include any special characters in your responses - this is
a voice conversation.`,
        },
      ],
    },
  };
  return vapiAssistant;
};
```

Додаток В

Серверна логіка роботи з AI-асистентами

```
'use server';

import {auth} from "@clerk/nextjs/server";
import {createSupabaseClient} from "@/lib/supabase";
import { revalidatePath } from "next/cache";

export const createCompanion = async (formData: CreateCompanion) => {
    const { userId: author } = await auth();
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('companions')
        .insert({...formData, author })
        .select();

    if(error || !data) throw new Error(error?.message || 'Failed to create a
companion');

    return data[0];
}

export const getAllCompanions = async ({
    limit = 10,
    page = 1,
    subject,
    topic
}: GetAllCompanions) => {
    const { userId } = await auth();
    const supabase = createSupabaseClient();

    let query = supabase.from('companions').select('*');

    if(subject && topic) {
        query = query
            .ilike('subject', `%${subject}%`)
            .or(`topic.ilike.%${topic}%,name.ilike.%${topic}%`)
    } else if(subject) {
        query = query.ilike('subject', `%${subject}%`)
    } else if(topic) {
        query = query.or(`topic.ilike.%${topic}%,name.ilike.%${topic}%`)
    }

    query = query.range((page - 1) * limit, page * limit - 1);

    const { data: companions, error } = await query;

    if(error) throw new Error(error.message);

    if (userId && companions) {
        const { data: bookmarks } = await supabase
            .from('bookmarks')
            .select('companion_id')
            .eq('user_id', userId);

        const bookmarkedIds = new Set(bookmarks?.map(b => b.companion_id) || []);

        return companions.map(c => ({
            ...c,
            bookmarked: bookmarkedIds.has(c.id)
```

Продовження додатку В

```
}));
    }

    return companions?.map(c => ({ ...c, bookmarked: false })) || [];
}

export const getCompanion = async (id: string) => {
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('companions')
        .select()
        .eq('id', id)
        .single();

    if(error) throw new Error(error.message);
    if(!data) throw new Error('Companion not found');

    return data;
}

export const addToSessionHistory = async (companionId: string) => {
    const { userId } = await auth();
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('session_history')
        .insert({
            companion_id: companionId,
            user_id: userId,
        })

    if(error) throw new Error(error.message);

    return data;
}

export const getRecentSessions = async (limit = 10) => {
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('session_history')
        .select(`companions:companion_id (*)`)
        .order('created_at', { ascending: false })
        .limit(limit)

    if(error) throw new Error(error.message);

    return data.map(({ companions }) => companions);
}

export const getUserSessions = async (userId: string, limit = 10) => {
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('session_history')
        .select(`companions:companion_id (*)`)
        .eq('user_id', userId)
        .order('created_at', { ascending: false })
        .limit(limit)
```

Продовження додатку В

```
    if(error) throw new Error(error.message);

    return data.map(({ companions }) => companions);
}

export const getUserCompanions = async (userId: string) => {
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from('companions')
        .select()
        .eq('author', userId)

    if(error) throw new Error(error.message);

    return data;
}

export const newCompanionPermissions = async () => {
    const { userId, has } = await auth();
    const supabase = createSupabaseClient();

    let limit = 0;

    if(has({ plan: 'pro' })) {
        return true;
    } else if(has({ feature: "3_companion_limit" })) {
        limit = 3;
    } else if(has({ feature: "10_companion_limit" })) {
        limit = 10;
    }

    const { data, error } = await supabase
        .from('companions')
        .select('id', { count: 'exact' })
        .eq('author', userId)

    if(error) throw new Error(error.message);

    const companionCount = data?.length;

    return companionCount < limit;
}

export const addBookmark = async (companionId: string, path: string) => {
    const { userId } = await auth();
    if (!userId) return;

    const supabase = createSupabaseClient();

    const { data: existing } = await supabase
        .from("bookmarks")
        .select("id")
        .eq("companion_id", companionId)
        .eq("user_id", userId)
        .maybeSingle();

    if (existing) {
        revalidatePath(path);
        return existing;
    }
```

Продовження додатку В

```
    const { data, error } = await supabase
        .from("bookmarks")
        .insert({
            companion_id: companionId,
            user_id: userId,
        })
        .select();

    if (error) {
        console.error("Error adding bookmark:", error.message);
        throw new Error(error.message);
    }

    revalidatePath(path);
    return data;
};

export const removeBookmark = async (companionId: string, path: string) => {
    const { userId } = await auth();
    if (!userId) return;

    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from("bookmarks")
        .delete()
        .eq("companion_id", companionId)
        .eq("user_id", userId);

    if (error) {
        console.error("Error removing bookmark:", error.message);
        throw new Error(error.message);
    }

    revalidatePath(path);
    return data;
};

export const getBookmarkedCompanions = async (userId: string) => {
    const supabase = createSupabaseClient();

    const { data, error } = await supabase
        .from("bookmarks")
        .select(`companions:companion_id (*)`)
        .eq("user_id", userId);

    if (error) throw new Error(error.message);

    return data.map(({ companions }) => companions);
};
```